Introduction

1 Introduction

1.1 Learn Game Design Patterns by Remixing a MakeCode Platformer

This course is built on the simple idea of starting with a platform game and adding familiar game patterns and features. The resources here are part of a learning model which has the following elements:

- Game Making MISSIONS main missions involve choosing authentic game making design patterns from a menu. Other side missions help with engagement of different making preferences.

- A MAP of Learning Outcomes particularly suited to digital game making, presented in an accessible format to teachers and learners.

- Design METHODS are techniques that facilitators and learners can use to help navigate the process of making a game and reflecting on what is being learned in the process.

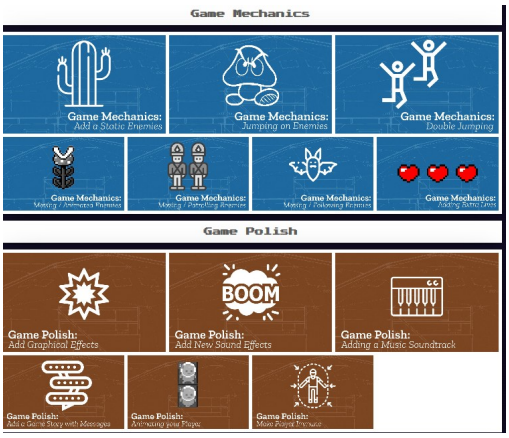
If you find mistakes in this resource or want to contribute please start an issue here.

1.1.1 Remix a Starting Game Template

A fun way to start coding and game making is to remix a starting template of a Platform Game by adding in different features which we call Game Patterns

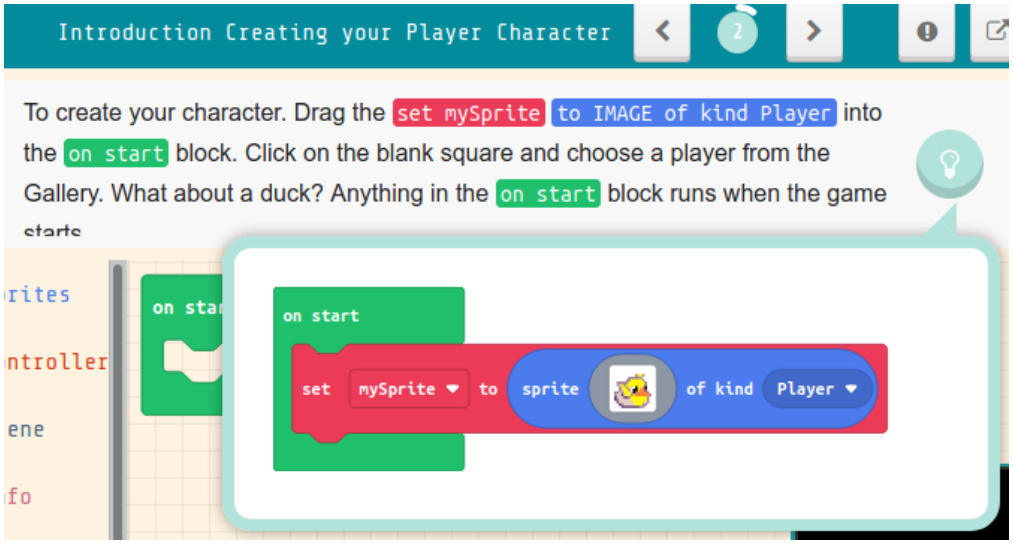
Starting Game To Remix

Patterns to Add to Game



1.1.2 Step by Step Instructions from First Principles

Another way to learn coding from first principles is to learn how to [make the sample game from scratch] and you can do that by following the following tutorials made with the MakeCode tutorial system.



tutorial

Part One - Create a Player character - Move the Player element around the screen - Create a Game Space using platforms using the tilemap tool - Add Food to collect using the tilemap tool

Part Two - Adding an end goal that you must touch to win game - Make it so you must collect all Food to win game - Creating a larger Game Space - Adding Levels - Adding a Timer

[follow the tutorial online here]

1.1.3 Follow an Example Course

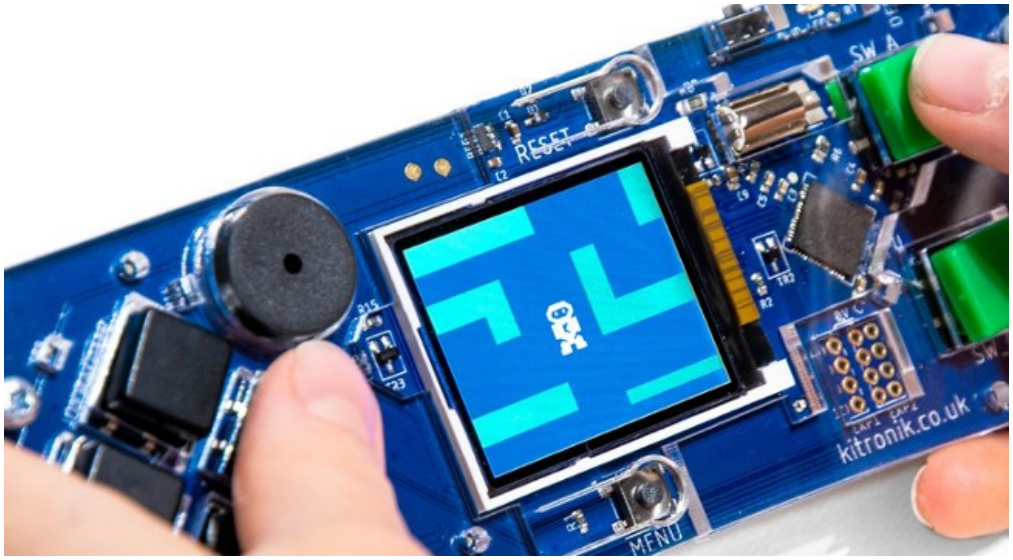
There is material here for a five session course on game making. * [Session Materials to run with a group of participants]

1.2 About the MakeCode Arcade Tool

makecode layout

This course uses MakeCode Arcade as its tool. There are several advantages to this tool.

- Like Scratch the block coding tool reduces chances of getting stuck with code syntax errors
- Unlike scratch it is created with arcade like games in mind and includes relevant tools like acceleration for gravity
- There are fun hand-held devices you can easily play your game on. Great for group playtesting.
- The support and making community is small but very friendly and responsive



a hand held device

The shape of this course could be applied to any game making platform. In fact there are similar courses available see Phaser Game Making (Javascript) and Making Serious Games in Scratch

1.3 Ideas behind this Course

This ethos of this course is broadly in-line with concepts of Project Based Learning. More specifically here's how some of those ideas come into focus with this material.

1.3.1 Learning via Hands-on Tinkering

This intention of this course is not to teach the underlying concepts of computer science from first principles. Here the idea is to build competency with the practicalities of using the coding tool. In an art studio students start by experimenting with materials and getting to know their tools first. Large concepts come later.

1.3.2 Playful Learning

Many sessions start with games and the overall ethos is to try to encourage an informal approach to learning which prioritises a positive connection to the making experience over absorbtion of testable curriculum knowledge.

1.3.3 Promoting Participant Choice and Self-Directed Learning

The use of a menu of Game Patterns as the hub of our learning allows and encourages the course participants to set their own goals and to monitor their own progress. It is also flexible enough to provide materials for those who want to learn concepts in a structured way although it priorities opportunities for more messy approaches to learning.

1.3.4 A Restricted Set of Learning Dimensions

To encourage messy, experimental learning many of the computer science concepts that it would be possible to teach are intentionally put into the background to prioritise a inclusive process. Care has been taken to reduce the number of concepts and patterns in these areas to a manageable amount.

In addition to these subject related learning dimentions, there are notes on process related learning dimensions outside of subject knowledge that may be useful to facilitators and parents. These include:

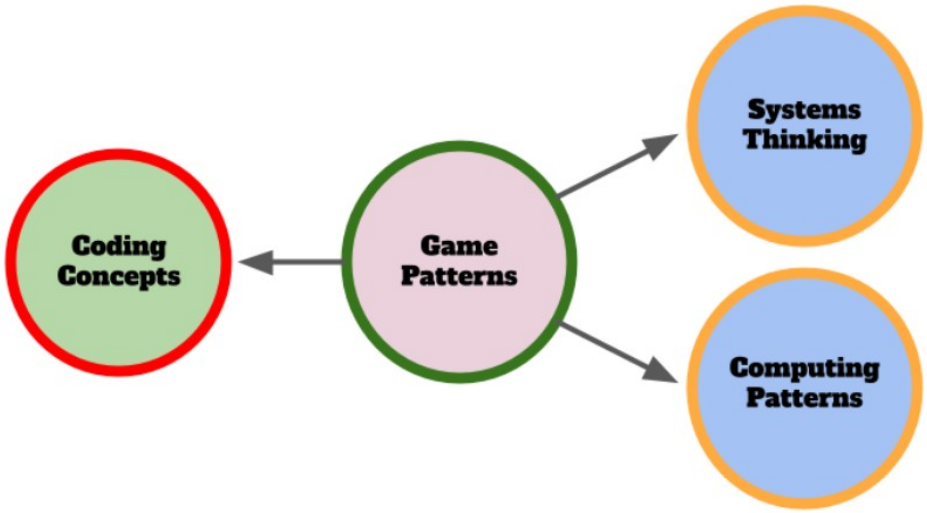
- Coding Concepts: mechanics of coding including using variables, loops, lists, logic, different types of data, input events and creating functions.
- Design Practices: including goal setting, creating, testing and reflecting.
- Systems Patterns: systems dynamics and different types of feedback loops.

1.3.5 Starting with the Familiar, Zoom into Detail, Zoom out to Wider Patterns

The starting point for participants of this course are the Game Patterns of a platform game which are likely to be familiar to them from exposure to family oriented video consoles or mobile games. Through the process of adding these patterns to their games, participants will come into contact with Coding Concepts needed to enact these features using the specific tools of the MakeCode system. In this way the **Zoom into the Detail** needed to make ideas into reality.

In the same process course participants also have the chance to connect the more immediate learning in the context of games to Wider Patterns of Computing and Systems. While this is not explicitly taught, parents and facilitators **Zoom out** to point out how the concepts and patterns

in Games can be applied in other fields, for example those of Interactive Media Design and Systems Thinking.



Wider and Narrower Patterns

1.4 Where to get started?

If you want to get started or begin to investigate what is available the following links will help you.

- View a graphical menu of Game Patterns which contains a short intro on how to apply them to the starting game template
- Get an overall understanding of the starting game template with the step-by-step tutorial
- Browse the Group Course contents or start to work your way through the individual online course
- Check out the potential Learning Dimensions of this project

2 Game Patterns

The key idea of this approach to game making is to start with a simple Platform Game Template to Remix and to add Game Patterns from a menu of possibilities. There are many ways of thinking about these patterns but in this guide we are dividing them up into the following:

mechanics space polish and systems

- Game Mechanics: things to do with the actions of the game
- Game Space: things to do with the layout of the game
- Game Polish: music, backgrounds, graphics and story elements
- Challenge and Systems: how different elements interact to create challenge

2.1 How to build your own game from the following Game Mechanics

The process of adding to the Platform Game Template to Remix. Start by playing the game and then clicking Edit Code to get started. Then pick one of the Game Patterns below and follow the instructions to add it to your game. Test it out and when you are happy with it, pick another and keep adding to it. The following tips will help you:

- Some Patterns need to be added before other ones will work so look at the related Game

Patterns section to see if there are other patterns you need to add first

- Keep records of the links to your published games as you progress in case you get stuck and need to go back one step
- Have a look at the activities of either the group course or an online individual course to enhance your learning
- Adding these Game Patterns creates learning not only of Coding Concepts but also Wider Computing Patterns and Systems Concepts

2.2 Game Mechanics

Add or change what you do in the game

- Add Player Lives
- Add Static Hazard
- Add an Animated Enemy
- Jump on Enemy to Zap them

- Double Jump
- Moving / Patrolling Enemies
- Moving / Following Enemies

2.3 Game Polish

Change the look and feel of the game and add to the story

- Add Graphical Effects
- Add Sound Effects
- Add a Sound Track (Music)
- Add a Game Story with Messages
- Animate your Player's Movements
- Make Player Immune

2.4 Game Space

Change the shape or nature of the playing space of the game.

- Change Design of Levels
- Add More Levels
- Change Shape of Levels
- Change the Background Image
- Key and Door

2.5 Challenge and Systems

Through challenge and systems, games get harder as you progress to keep your interest.

- Gain Points when Collecting Food
- Add a Timer
- Collect all Food before Progressing
- Power up - Higher Jump
- Power up - Player Speed
- Random Doubling Enemies

3 Learning Dimensions of this Project

This section includes a map of different learning dimensions which are possible and likely to emerge from taking part in this game making course.

It is inspired by the work of Bevan and Petrich around their study of ‘tinkering’ in science museums, who through close observation in partnership with learning facilitators mapped some of the complex learning processes which may be hard to spot in a quite chaotic and messy learning environment.

Other ideas behind the structure of this section are explored here.

- Coding Concepts
- Systems Patterns
- Design Practices

3.1 Coding Concepts

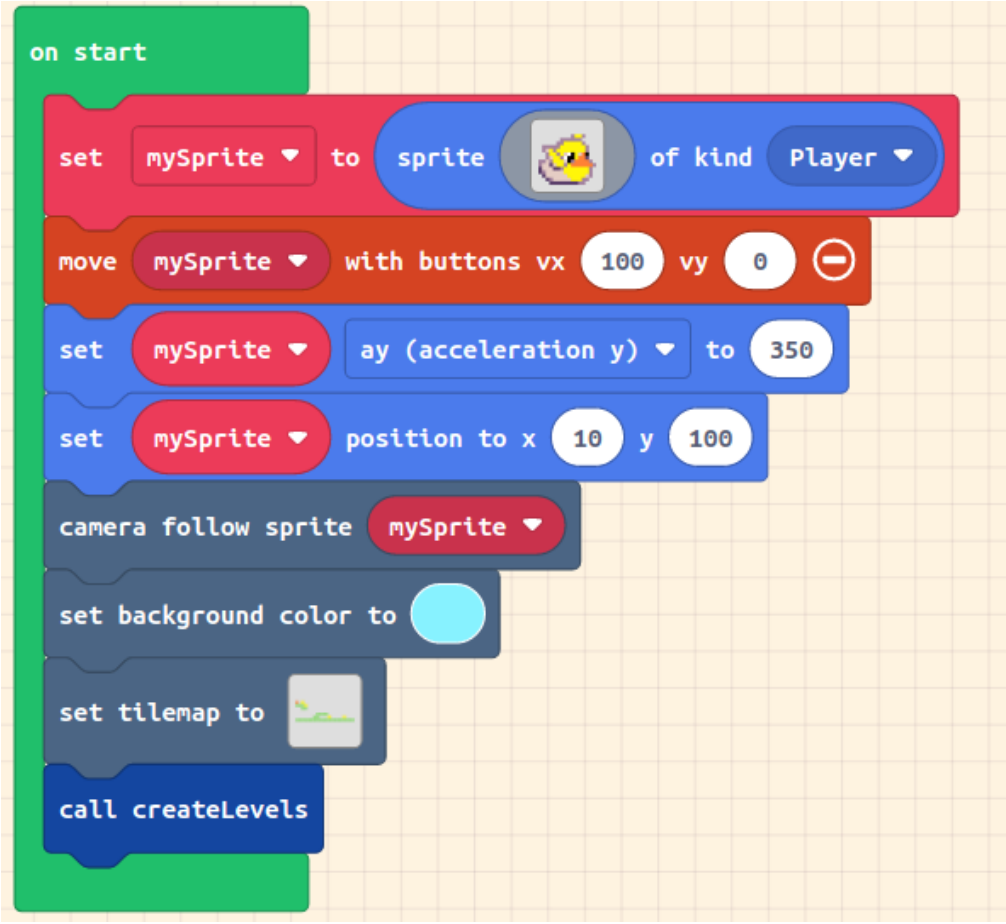
These concepts are needed to put Algorithmic Thinking into practice. The following are loosely based on the computational thinking concepts of Brennan and Resnick.

- Sequences
- Variables
- Logic
- Loops
- Arrays
- Creating Functions
- Change Listener
- Input Event

3.1.1 Sequences

- **Description:** Tasks that can be expressed as a series of steps.

- **Why is it needed :** Computers don't have common sense. They need precise instructions that are broken down into individual steps. The order that tasks happen in is significant. Some bugs are due to errors to commands being run in the wrong sequence.
- **How it happens in practice:** You would put the sequence inside a on start block. An example from our starting template is below.



logic

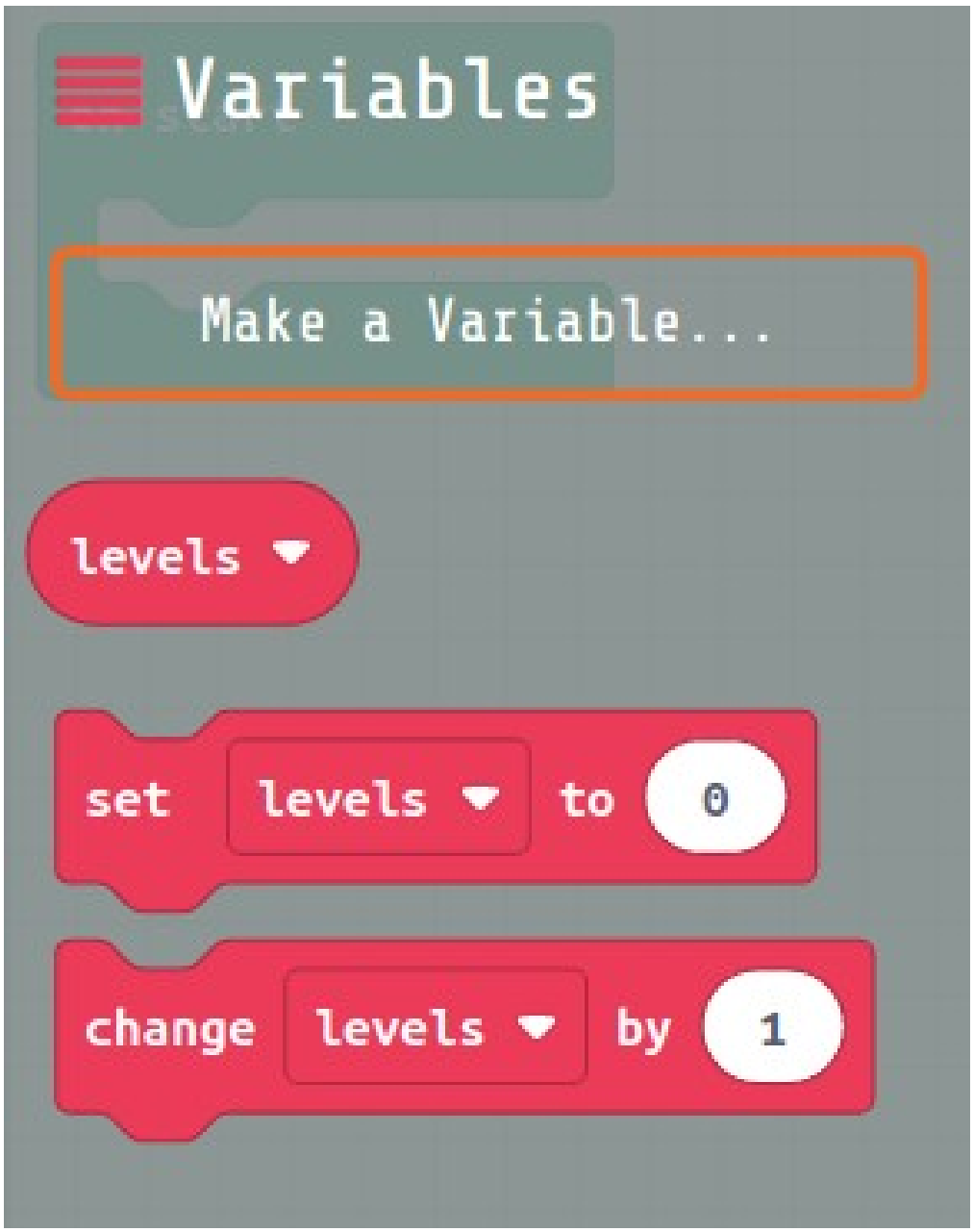
A common practice when there is a set of instructions that might be run in more than one situation, or to keep code neat is to create a separate function. The last block of the code above does that.

- **Examples in our Game Patterns:** Add Player Lives,

3.1.2 Variables

- **Description:** Variables are a kind of data that allow storing, retrieving, and updating values.

- **Why is it needed :** Variables are useful when we know something may change when our program is running. For example if we create more than one level, we can create a variable called level and increase this each time a player progresses.
- **How it happens in practice:** You can create and set variable us using blocks in the Variable section.

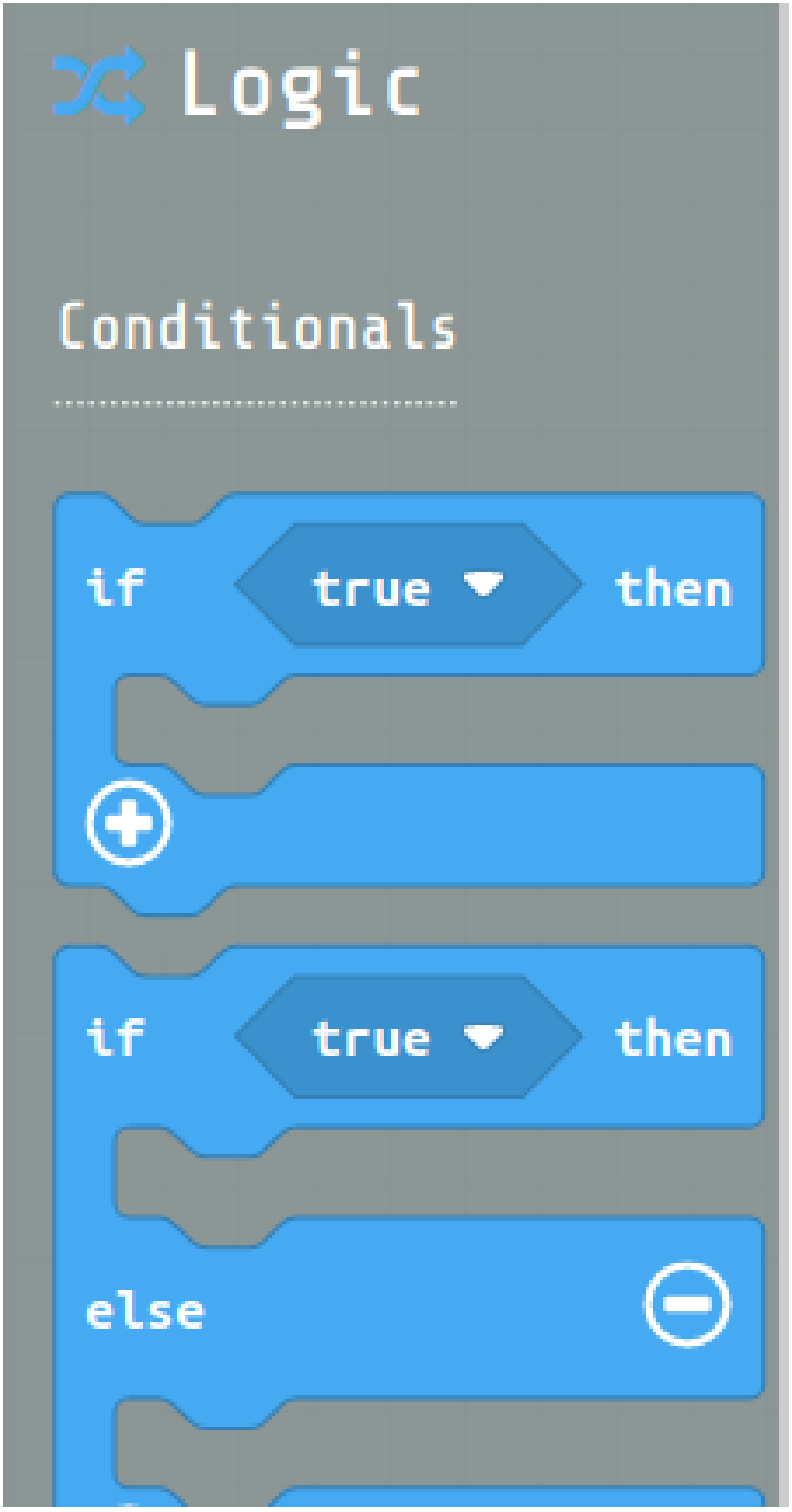


logic

- **Examples in our Game Patterns:** Add More Levels,

3.1.3 Logic

- **Description:** Logic (Conditionals) give the ability to make decisions based on certain conditions so that of multiple outcomes may occur.
- **Why is it needed :** Logic / Conditionals are a key concept in interactive media as it allows for different things to happen based on different input choices from the user / player. Or in games to respond to different conditions of play in the game.
- **How it happens in practice:** There are logic blocks in MakeCode which express different pathways using the if / then / else logical pattern. Comparison blocks let you compare the values of mathematical and text values and then run different blocks of code depending on the result.


Logic
Conditionals
if
true
then
if
true
then
else

logic

- **Examples in our Game Patterns:** Animate your Player's Movements

3.1.4 Loops

- **Description:** A loop is a way of repeating a piece of code so that it runs more than once.
- **Why is it needed :** We could programme an enemy to move 100 pixels to the right, wait 0.2 seconds, and then to repeat the action – moving another 10 steps, and waiting another 0.2 seconds. What if, instead of a single repetition of the action, we want the enemy to move and wait three more times? We could easily add more move and wait blocks. But what if we wanted to repeat the process 100 or 1000 more times? Loops are a way to run the same sequence many times.
- **How it happens in practice:** in Make Code we use green Loops blocks to repeat code. An example is the "repeat x times" block.

loop one

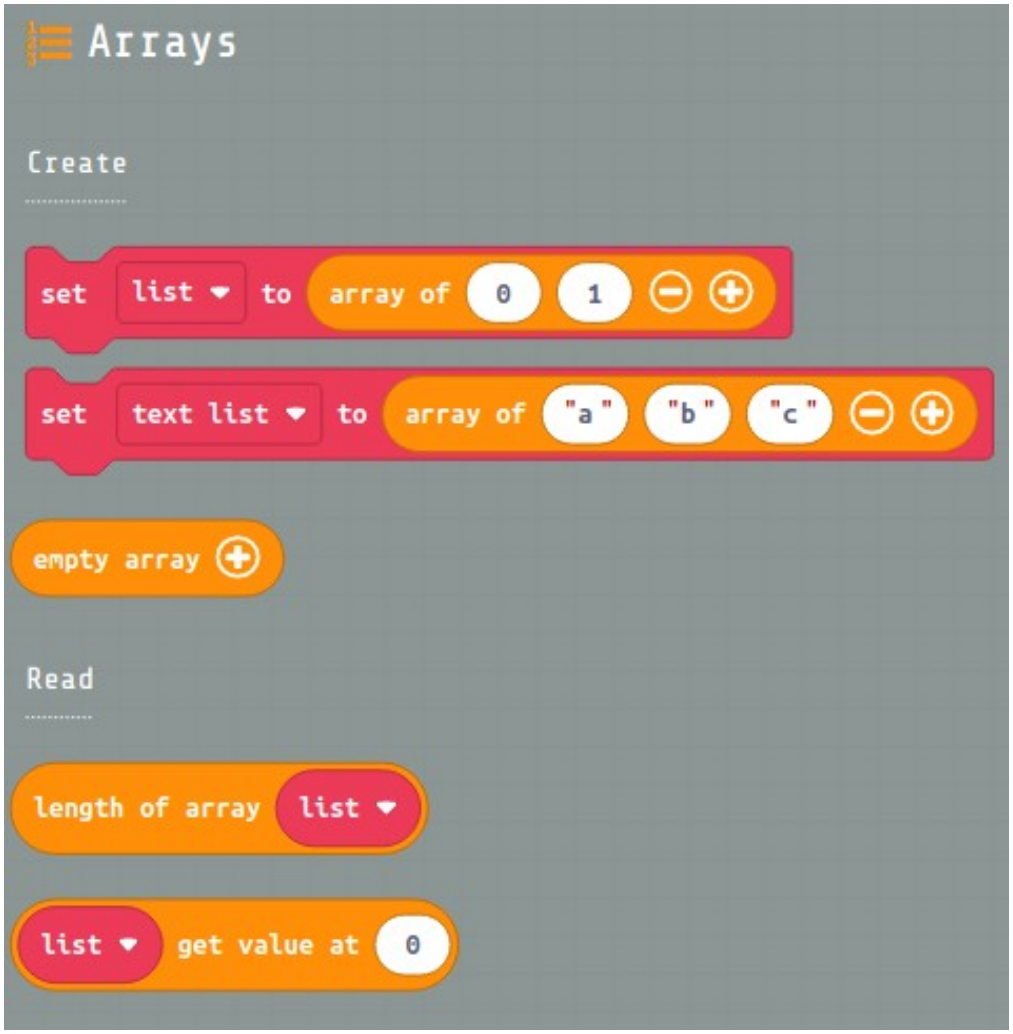
- **Examples in our Game Patterns:** Add Static Enemies,

3.1.5 Arrays

- **Description:** Arrays are a kind of list which also allow you to store and retrieve values but

as a flexible collection of numbers or strings or images.

- **Why is it needed :** Arrays are commonly used in combination with loops, to loop through a list of things to repeat a process. An example would be the way a tile map is turned into a game layout.
- **How it happens in practice:** There is a group of code blocks in the advanced section of the Arcade MakeCode interface which help us to work with Arrays. Also using tilemaps to represent level layout is a kind of array.



array

- **Examples in our Game Patterns:** Change Design of Levels

3.1.6 Creating Functions

- **Description:** Also known as abstraction and decompostion (by modularisation), this is a

pattern that allows programmes to be broken down into smaller pieces, which do a specialist job. In this case the pieces are called functions.

- **Why is it needed :** The concept is in line with a programming principle called DRY (Don't Repeat Yourself). It allows you to call the same piece of code from different parts of your programme so you don't have to repeat them in more than one place.
- **How it happens in practice:** Examples include: - creating levels on start and when goal is reached
- **Examples in our Game Patterns:** Add Levels

3.1.7 Events

Events occur at different times in the running of a computer programme. Two types are User Input Events and Change Listeners.

Input Event

- **Description:** Also known as an event handler. Events – one thing causing another thing to happen – are important to interactive media like games. For example, a button triggering the start of play, or controlling game player' s movements.
- **Why is it needed :** Working with User Input is a key part of most computer programming design, it allows the user to interact with the programme to make things happen.
- **How it happens in practice:** Examples include:
  - controlling the player movements on screen
  - responding to button presses for jumping for firing etc

Change Listener

- **Description:** Also known as States, this pattern is useful when you want one part of your programme to listen out for changes in the stage of another. An example would be listening for an overlap between the player and an enemy. The programme can then take

action when this overlap change happens or a particular condition is true.

- **Why is it needed :** In many media programmes you need to change formatting or to make something happen based on the conditions of other objects.
- **How it happens in practice:** Examples include:
  - listening for the player being in a condition of overlap with an Enemy
  - changing the animation of a player based on if they are still, moving or jumping

3.2 Systems Patterns

- Systems Elements
- Systems Dynamics
- Reinforcing Feedback Loops
- Balancing Feedback Loops

3.2.1 Systems Elements

- **Description:** This concepts allows you to break down a system into its different parts. In our game we can break down the into game components, goal, space, rules and mechanics. This helps to start to identify the different kinds of links between the system elements.
- **Why is it needed :** This process of recognising and naming the system elements is the start of being able to understand how a system works.
- **How it happens in practice:** Examples include:
  - naming the components, rules, space, mechanics and goals of our starting platformer
  - the process of making changes to these game elements

3.2.2 Systems Dynamics

- **Description:** Links between systems elements can be static or dynamic. The idea of stocks and flows is important. For example in a game you can have a static number of

hazards that never changes, or a dynamic population of enemies that may increase or decrease. The nature of relationship between enemies and player will then be different.

- **Why is it needed :** Recognising dynamic parts of game system allows you to make more informed decisions about how it will change and react to changes. For example understanding these dynamics are important when making a game to make sure the level of challenge for the player is suitable.
- **How it happens in practice:** Examples include:
  - increasing the number of enemies to increase challenge

3.2.3 Reinforcing Feedback Loops

- **Description:** Feedback loops can be balancing or reinforcing. We can thing of a reinforcing loop as change that gets progressively either bigger or smaller. For example a population of rabbits spiralling larger and larger in a place where they have no natural predators and plenty of food.
- **Why is it needed :** In a game you may want a reinforcing loop if you want to increase the challenge for a game. For example, you may want the number of enemies to drastically increase towards the end of a level.
- **How it happens in practice:** Examples include:
  - creating a hectic game with many spawning enemies

3.2.4 Balancing Feedback Loops

- **Description:** Feedback loops can be balancing or reinforcing. Balancing feedback loops tend to react to changes in a system with a responding change that brings balance back.
- **Why is it needed:** In eco-systems balancing loops work to keep a system stable, preventing there from being too many predators for example, as they will die off if there isn’t enough food for them to prey on. In games balance is needed to maintain the right level of challenge for the player.

- **How it happens in practice:** Examples include:
    - gravity acting as a balancing force to jump velocity
    - keeping a check on the number of enemies that the player has to face so that the game doesn't become impossible

3.3 Design Practices

3.3.1 Defining Design Practices

This mapping of design practices encompasses concepts from the following frameworks.

- Design Thinking Skills: This computational thinking definition of Brennan and Resnick contains a section on design practices afforded by the Scratch media authoring tool. His creative design spiral is similar.
- 21st Century Skills: There certain elements of key 21st Century Skills like communication, collaboration, critical thinking and creativity that are key to design practices.
- Digital and web literacy: The web literacy project by Mozilla maps maps digital skills suited to collaborative and digital game making, some element are included here.

The whole process is inspired by Bevan and all's learning dimensions framework for making and tinkering. The following learning dimensions have emerged having been selected based on observation the practices being used in the game making programs:

- Goal Setting
- Being Incremental and Iterative
- Developing Shared Vocabulary
- Reusing and Remixing
- Web Navigation
- Problem Solving
- Version Control
- Debugging
- Reusing and Remixing

- Game Testing and Publishing

3.3.2 Goal Setting

- **Description:** The process of remixing a game involves decisions to change the game intentionally in minor or more fundamental ways. These moments of participant initiative draw from and build on existing knowledge.
- **How it happens in practice:** Example activities which demonstrate this include:
  - Invitation to peers to set new goal or revise existing goals
  - Participants expressing new goals to include in game or around the wider making experience
  - Clarification or questions about desired game design behaviour

3.3.3 Being Incremental and Iterative

- **Description:** Designing a project is not a clean, sequential process of first identifying a concept for a project, then developing a plan for the design, and then implementing the design in code. It is an adaptive process, one in which the plan might change in response to approaching a solution in small steps.
- **How it happens in practice:** Examples include:
  - Small revisions to address the game pattern being worked on or sub goals
  - Changes which seeking to maintain or create game balance for suitable challenge level
  - Evaluating aesthetics of graphics / sound elements and revising further

3.3.4 Developing Shared Vocabulary

- **Description:** The process of working on a joint project involves the development of shared understanding of the problem being worked on. This may involve checking working concepts by advancing them and questioning meaning of other's expression. Related activities also include

- **How it happens in practice:** Examples include:
    - Asking or providing clarification of terms being used
    - Requesting or offering help in solving problems using specialist terms

3.3.5 Collaborative Production

- **Description:** Where participants work together on game making as pairs or in family groups participants often demonstrate behaviour which encourage collaboration and participation in the broader community aspects of game making. The process of offering, providing or requesting help with the non-technical aspects of advancing the project in question. There are other aspects of support which are needed outside of the session like access to resources or helping participants to access learning in a general sense. This is often the role of parents but sometimes siblings or child participants step into this role.
- **How it happens in practice:** Example activities which demonstrate this include:
    - Redirection of other participant's activity back to reflect their earlier goals
    - Invitation to imagine the end game player's experience
    - Advancing or suggesting alterations to collaborative working practices
    - Emotional support through general encouragement and keeping a sense of perspective
    - Brokering access to wider learning opportunities

3.3.6 Reusing and Remixing

- **Description:** Building on other people's work has been a long-standing practice in programming, and has only been amplified by network technologies that provide access to a wide range of other people's work to reuse and remix. This behaviour may be remixing the work of others or through helping others to replicate their own work.
- **How it happens in practice:** Example activities which demonstrate this include:
    - Direct copying of game features from other games by replicating code
    - More indirect copying of features and ideas via observation or conversation with

peers

3.3.7 Web Navigation

- **Description:** The detail of moving from website to another, especially when keeping several browser tabs open, offers the
  chance to learn and share a variety of practices which make up key web-navigation literacies. This is an area where young people often are able to help adults.
- **How it happens in practice:** Activities include:
  - Asking or offering help in navigating between websites or browser tabs
  - Parents or siblings as a repository for the logins and passwords of their children

3.3.8 Problem Solving

- **Description:** Problem solving and developing understanding develops some of the aspects of collaborative work to working processes or domain knowledge.
- **How it happens in practice:** Example behaviour includes:
  - Giving explanation for outcomes or tactics
  - Applying existing knowledge to the problem at hand
  - Striving to understand project concepts
  - Linking project experience to wider learning

3.3.9 Version Control

- **Description:** The process of keeping a track of different versions of your work is called version control. There are many ways of doing it but doing it in a manually to keep track of changes to our game is a great way to start to understand the value of this process. The way MakeCode doesn’t support log ins encourages a process of saving versions of your work frequently and saving them in a document, say an online google doc with dates and descriptions.

- **How it happens in practice:** Example behaviour showing this skill includes:
  - Encouragement between participants to save work
  - Keeping written records of changes
  - Deciding to create a new version of work
  - Explaining and reminding peers of value of version control

3.3.10 Debugging

- **Description:** Various testing and debugging practices are possible from learning to scan block code for error messages, to using tools to watch variables.
- **How it happens in practice:** Debugging involves different strategies for dealing with different kinds of errors including:
  - Glitches - code still works but unexpected result (questioning
  - No errors blank screen, no idea (often retrace steps, go back to earlier version, or have to start again)
  - Clues, greyed out code, red dots, error messages (try to narrow down issue, remove sections of code, again go back, try to find duplication of code blocks)

3.3.11 Game Testing and Publishing

- **Description:** Game testing can be of one's own game or as 'playtesting' of playing each other's game and giving and receiving feedback on game play. The process of making work public or sharing a 'finished' version with the group can support development of the evaluation.
- **How it happens in practice:** Example activities which demonstrate this includes:
  - Expression of pride, or fun when playing own game
  - Imagining the user's experience of playing their game
  - Giving or receiving feedback either spontaneously or with support form a feedback / playtesting sheet
  - Creating a public link from a game making workspace

4 Methods

The resources here are part of a learning model which has the following elements:

- Game Making MISSIONS main missions involve choosing authentic game making design patterns from a menu. Other side missions help with engagement of different making preferences.
- A MAP of Learning Outcomes particularly suited to digital game making, presented in an accessible format to teachers and learners.

- Design METHODS are techniques that facilitators and learners can use to help navigate the process of making a game and reflecting on what is being learned in the process.

This page outlines methods and groups them by theme.

Methods Using Missions

- Macro Mission - Tell a Systems Story (environmental or social) through a game

- Discovering Game Mechanics
- Improving Half-Baked Game
- Limits to the Mission based approach - meeting your self in the middle
- Player and Maker types & Specialised missions

Methods Using the Learning Map / Design Process

- Circle / Physical Reflection Games:
- Guidance on Running Creative Design Sessions: imagined audience,

Other Methods

- Using Hardware and Playtesting to focus on Imagined Audience

- Predicting Code Outcomes via Games

- Coding Concepts via step by step tutorials
- Supporting Debugging

4.2 Discovering Game Mechanics

Knowing what the main mechanics for games are and making sure that they feel fun and responsive for our player is key to making a great game.

Game Mechanics are often represented as verbs. Examples include;

- Collecting
- Avoiding
- Chasing
- Shooting
- Walking / Running

The process of identifying the verbs of the game, also normally allows the participants to list the main components of the game too. This can start to build a knowledge of systems dynamics.

4.2.1 Identify Game Mechanics in Existing Games

We will start to understand games by playing a few arcade games and starting to look at what makes them tick. On entry participants play the following games (from MakeCode)

- Chicken Run
- Hot Air Ballon Game
- Bunny Hop
- Eat the Fruit
- Level Up

Write down the main VERBs for these games. Some have more than one Mechanic.

Extending this activity: This activity can be extended to analyse games using more game Components in line with game theory. A hand out is available here.

4.3 Improving Half Baked Games

This process is a way to jump right into altering code. Instead of planning from first principles, learners are given a game that is designed to provoke players to tweak it. This process is inspired by game modding and a process called using half-baked games is often called

4.3.1 Example Activity - Fix the Broken Game

Now we will start coding by jumping right in. Try to play this game. It's broken right? You can't jump high enough. Now look at the code of this broken game and use the link to workshop cards with activities to fix the game and add some elements quickly.

- Broken Platform Game to Fix
- Supporting Activity Cards

At the end of the session make sure to Publish a copy of your game and to give it your own name. Keep a record of the link to your game. We'll come back to it next week.

4.4 Circle / Physical / Drama / Reflection Games

From stables of drama and community process. They are to introduce concepts in a playful way, and to help reflection - often this works just to draw participants away from their screens to interactive in a different way.

4.4.1 Example Activity - Reflection Web Activity to End a Session

Play a quick reflection game to find out what people thought and got out of the first session. The Reflection Web is a fun one to start with. Ask the following questions.

- Share something you found fun
- Share something you found hard
- Share something that surprised you

4.4.2 Line Up Reflection Activity

Paper Slide: On paper, individuals or groups sketch and write what they learned or have been working on this session. Then line up and, one at a time, slide their work under a video camera while quickly summarizing what was learned. The camera doesn't stop recording until each representative has completed his or her summary.

For this reflection activity ask participants to include any new objectives or work on systems dynamics.

4.4.3 Example Activity - A Game and Tips to Keep track of our Progress and our Games

Play musical chairs. When someone is left standing up. Ask themselves

- what new idea or coding technique jumped out for you today?
- what do you want to do next?

Then share the following tips for participants to make the most of this course:

- Working on your games at home is a great way to make the most of this course - check that you have all you need to be able to access your game later
- Be careful in keeping track of the location of different versions of your games, include a version number, describe what changed, even include a bit about what you are learned since the last version
- If this works well then why not keep a separate learning journal to reflect on what you are learning?

4.4.4 Example Activity - Bomb and Shield

Play a game to understand the elements of a game. There is a game called bomb and shield which you can play with a small group in the following way.

- Mark out your game space, no-one can walk or run outside of that.
- Everyone choose one person but don't say who it is. This person i your bomb

- Choose another person but don’t say who it is. This person i your shield
- Everyone walk around the space but try to keep your shield between you and tyour bomb.

Say Freeze. Stop the game and see who is safe. Play again, get them to swap people or choose new people.

Exploring the Elements of a Game

Outline the different elements of a game which can be changed by the designer to change the nature of the game.

- Components - the nouns of the game for example for a platformer (player, collectable food, enemies, end goal sprite,)
- Mechanics - the verbs of the game (walking, jumping, collecting)
- Space - changes to the layout of the game (location of platforms, does the space scroll when you move)
- Rules - what are you allowed to do, not allowed to do, what must you do (e.g. you must collect all food before progressing a level)

- Goal - what is the main goal of the player of your game (e.g. collecting points or progress to levels by reaching an end goal?)

Have these written out and ask participants to try to match them with the following elements from our game.

- Bomb, Shield, Player
- A clear space outside or in which is about the size of a room
- You are not allowed to push people out the way.
- Win the game by staying safe by having your shield protect you from your bomb
- Walking

Explain that digital games can be analysed in the same way and that we will learn to use this knowledge to make a challenging game.

4.4.5 Fun Project Evaluation and Reflection

You may want to use this as an opportunity to gather feedback from participants on the course to make improvements and to encourage final reflections from course members on what they gained from the game making activities. The following questions may be useful.

FUN THIS UP?

- How has your learning journey been?
- What is the most noticeable change along the way?
- Can you share about your changes in confidence in use of the specifics of coding concepts?
- What do you feel you've learned about wider patterns - computer programming/ systems designs?
- What are your final takeaways?

4.5 Drawing on Home and Game Cultures

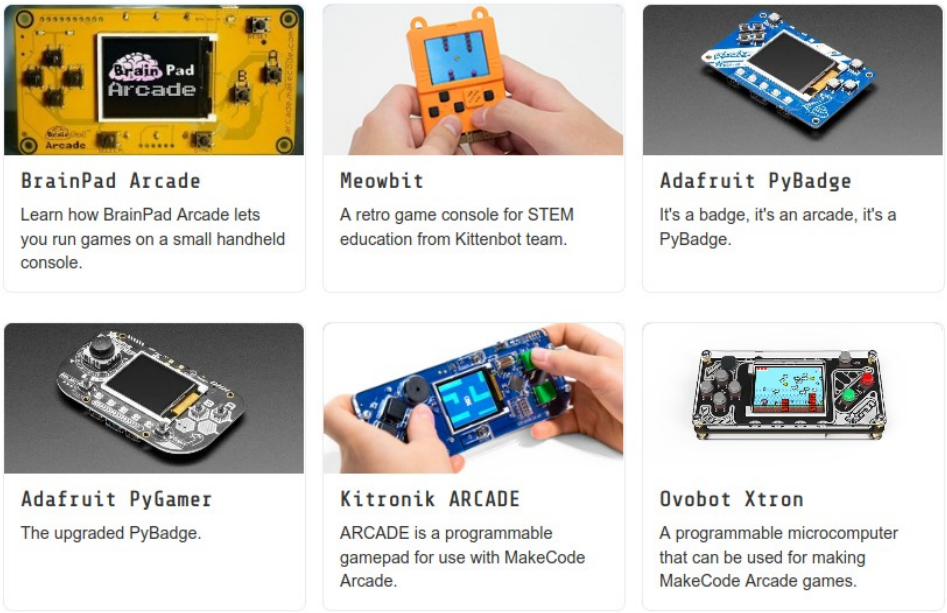
This method is inspired by third space theory and the importance of being able to work with Home Discourses.

4.5.1 Using Hardware Device to Increase Motivation

There are different hardware available for making a splash with the finished games. Using hardware with buttons can also increase the playability of the games as well.

These include:

Hand-held Devices: A number of game-boy like devices exist which are great to use with old school buttons and mini screens. See a list here.



boxes

Arcade Cabinets: A bit of carpentry can create a fun looking arcade cabinet, we have some great ones where we pop in old laptops and use arcade buttons and [makey makey's to create great looking machines](#).

boxes

Cardboard Cabinets: Even more fun perhaps is to make your own arcade cabinet machine out of cardboard and hook up your buttons and makey makey's to that. [More info here](#)

boxes

You may want to run an extra session on making a console with computer electronics including hardware elements of arcade buttons. There are links to resources to help you here.

If your project involves cardboard elements then it can be a great idea for participants to take the cardboard home and to decorate it with ideas from their game.

4.5.2 Showcasing to and External / Authentic Audience

If you do have a chance it can be a really exciting experience to showcase your games at an event or in a public place where you can **get passing strangers to play your games**. For our courses at the Manchester Met University we have a captive audience of undergraduate students.

I have found that this is often where parents come into their element. **Ask parents to be cproactive to draw in passing people** to play and give feedback on games. **It is very satisfying to witness strangers play your game** and experience the kind of fun frustration and desire to try again to beat a particular part of your level design.

If this is not possible then try to make the final playing of each other's games a real event. **The use of the hardware and especially decoration of cardboard cabinets can really help to create**

a sense of an event. Also ask participants what they want to do to **celebrate, share food, play music** as a shared play list for example.

You may want to have in place a form which allows participants to **gather player reactions**. This can be via a written form or a quick recorded interview if suitable.

4.5.3 Exploring - What kind of Game Player / Maker are You?

As digital and online games became more complicated Richard Bartle proposed that different people play these games in different ways and look to get different things out of them. In short there are different kinds of players. The Bartle test finds out what kind of game player you are. **Do the online Bartle Player test.** You may be able to find a way of doing it as a group by moving around the room.

This is also true of the way that people play games. There's a well known model of different play style types by Richard Bartle. This model, which was based on observing and analysing the behaviours people playing together in a multi-user game, holds that there are four different kinds of play style interests, each of which is given a descriptive name: Griefers, Achievers, Explorers, and Socializers.

- Griefers: interfere with the functioning of the game world or the play experience of other players
- Achievers: accumulate status tokens by beating the rules-based challenges of the game world
- Explorers: discover the systems governing the operation of the game world
- Socializers: form relationships with other players by telling stories within the game world

Different kinds of games suit different play styles. One of the notable successes of recent years have been open world games that allow you to choose how you play the game. If you want to stick to the main missions you can follow guidance to do that but if you just want to explore or be social or mess around you have the chance to to do that too.

In the same way there are different styles of making games. I'm proposing the following;

- Social makers: form relationships with other game makers and players by finding out more about their work and telling stories in their game
- Planners: like to study to get a full knowledge of the tools and what is possible before they build up their game step-by-step
- Magpie makers: like trying out lots of different things and happy to borrow code, images and sound from anywhere for quick results
- Glitchers: mess around with the code trying to see if they can break it interesting ways and cause a bit of havoc

As well as different game player types, there are also different game maker types. These are listed here, but if you are a planner then you may want to really know how all of the code for our game works before you start to make other changes to it.

4.5.4 Step by Step Tutorials - (suits Planners especially)

If you are a planner then you may want to take the time to work your way through these tutorials. If not jump right on to the next section.

- Tutorial Part One
- Tutorial Part Two

4.5.5 Adding Game Patterns to a Starting Template (suits Magpies especially)

In part one we made changes by adding features called Game Patterns to a broken game. We are now going to start from with a similar Platform Game Template to Remix. It is a bit simpler, for example there is only one level and you don't have to collect all food before winning. There are many ways of thinking about these patterns but in this guide we are dividing them up into the following:

mechanics space polish and systems

- Game Mechanics: things to do with the actions of the game
- Game Space: things to do with the layout of the game
- Game Polish: music, backgrounds, graphics and story elements
- Challenge and Systems: how different elements interact to create challenge

Find a fuller list of the different Game Patterns and how to apply them on this Page.

4.5.6 Extra Missions (suits Socializers and Griefers especially)

Extra Mission Cards These extra missions encourage us to think of our games as dynamic systems that change as we play the game.

Mission Cards for this Sessions

You can point them towards elements of systems thinking that arise from these challenges. These may include the following.

- Systems Elements
- Systems Dynamics
- Balancing Feedback Loops
- Reinforcing Feedback Loops

4.6 Predicting Code Outcomes via a Matching Game

4.6.1 Matching Games to Code Game

On entry participants play the following games (from MakeCode)

- Galaga
- Duck Run
- Eat Fruit
- Level Up

The next activity is to try to match cut out screen grabs to each of the games. [See print outs here] (https://drive.google.com/drive/u/0/folders/1NEh-YHlNO_yr7lBx1tVySml-ZkvDtpbp).

Additionally as a bonus can you see any of the following:

- Variables
- Loops
- Logic
- Events

4.7 Guidance on Running Creative Design Sessions

4.7.1 Creative Design Game Making Session Summary

Follow the Creative Design Game Making Session pattern as outlined in week two.

- Set goals to add chosen game patterns
- Follow the coding help to make the changes you need
- Test the game yourself imagining the end player’s experi#### ence
- Play test other people’s game and get feedback for yours
- Reflect on our progress

4.7.2 Creative Design Checklist

Follow the Creative Design Game Making Session pattern as outlined in week two - **Set goals, follow the coding help, self-test the game yourself, play-testing, reflection on our progress**

In terms of goal setting. What is becoming special about your game. What are the final changes you want to make? Use the following check list to make quick changes to your game.

- Do you have a strong story for your game? Can you add one?
- Have you applied a Mission Card or a system pattern yet?
- Have you got a balanced of game patterns added?

Also pay special attention to the **play testing phase** this time to see if you can see any glitches that others may find in your game.

4.7.3 More Detail on Creative Design Game Making Sessions

Locate Academically. The sessions are informal and different participants will be doing different things at different times. However there is a general pattern to help organise the chaos of our game making sessions. The sessions are based around the following pattern.

Set Goals Have a look at the list of the different Game Patterns. Each Game Pattern has it's own page. with descriptions of the patterns and details of how to add it to your platform game. For now choose 2 patterns to add to your game. Choose at least one that you think will be quick to implement. We will add them to our game before the end of the session. Fill out the Goal Setting sheet. (Create this sheet)

Create See the sheet and the pages to help you make the changes. These are normally step by step instructions. Sometimes there are places where you make your own decisions. Think about story of the game and the characters involved.

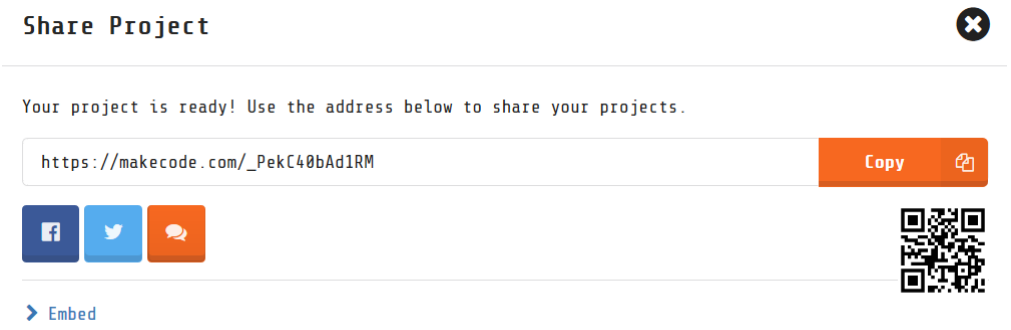
As we make changes to our game we will learn about the specific coding concepts needed to do computer programming, ideas like loops and logic. We'll also find links to wider patterns of computer programming and systems ideas that are important in the world of Human Computer Interaction and Systems Thinking. Have a look at a list of the learning dimensions we might pick

up through our game making.

Test You should play your game after every change you make to see how it has changed the playing experience. Keep in mind the experience of the end player when you are testing your game. Also try to think about the right level of challenge for your game.

Share with Playtesting Playtesting is a great way to really understand how the changes to your game make an impact on how the player experiences it. If you are doing Playtesting as a group then be sure to test a couple of games and make sure at least a couple of people test your game. Use this linked feedback sheet to help give feedback on games. << ADD SHEET

Publish Be sure to publish your work using the share link in the top right of the MakeCode edit screen. Give your game a name and ideally a version name. A good name communicates what's fun about your game or its story to your audience. You can also use the MakeCode forum as a way of getting feedback this. To do this click on the image of the speech bubbles.



mechanics space polish and systems

Reflect Reflecting on our progress allows us to getting the most out of the making to deepen our learning and make links to other areas of making.

There are different ways to do this, and if you are in a group you may want to make this into some kind of fun activity. There are some nice closure ideas here

4.7.4 Revising your Design Goals in the Final Stages

When you are making a game it is common to not have the time to make all the changes you want to make.

If you are nearing the end of the time you have allocated. You now have to make the difficult decisions to decide on the last big changes you will make to your game.

You also need to decide on which goals you will not be able to add to this game. To do this you can make a list and choose only the top two or three to work on. Try to use your experience from previous sessions to estimate how long this will take you. Try not to much more to your game in terms of new patterns. Focus as much as you can on the playability of your game. Make sure to allow time for a lot of self testing.

4.8 Supporting Debugging

4.8.1 Describe and Post a problem to your peers or the MakeCode forum

If you get stuck trying to add one element to your game then share a link of the broken game with your facilitator or the friendly online MakeCode community.

Also if you get stuck on one problem don't let that stop you. Go back to your last working version and try to add a different game pattern. That way when you solve your original problem you can add in the changes to your new version.

4.8.2 Understanding types of errors and dealing with them

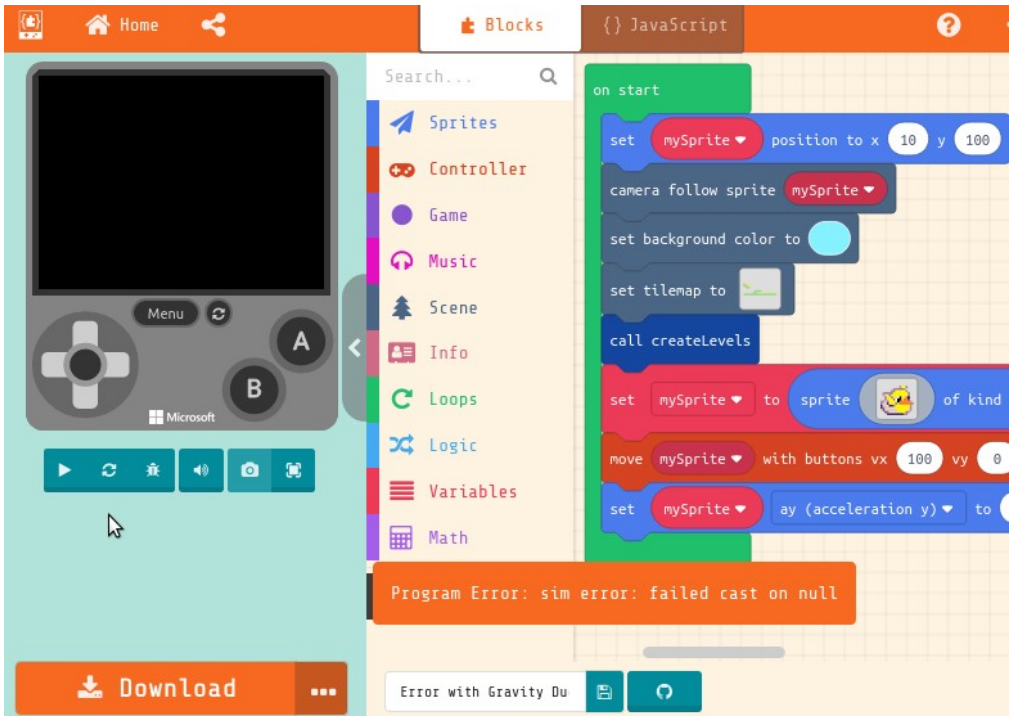
Can this be made into a game of some kind ? There are different kinds of that you may come up against. Knowing how to deal with them is a good tactic to keep us progressing.

- **Syntax and Program Errors** are errors in your code which stop the game from functioning at all.
- **No Behaviour Bugs** are errors which in your code which don't stop the game from functioning but your intended effect is not present when it should be.
- **Glitches** don't stop your game from running but as you play you see that there is an unintended effect the game does something different from what we want it to do.

Recognising and Fixing Program Errors and Syntax Errors

Syntax Errors are mistakes in the code we write, there are a lot of things MakeCode does to make it harder for us to make these kinds of syntax errors. However they can arise when we put the wrong kind of block in a particular location.

You may get a **Program Error** on the screen and a Black screen in the preview window.



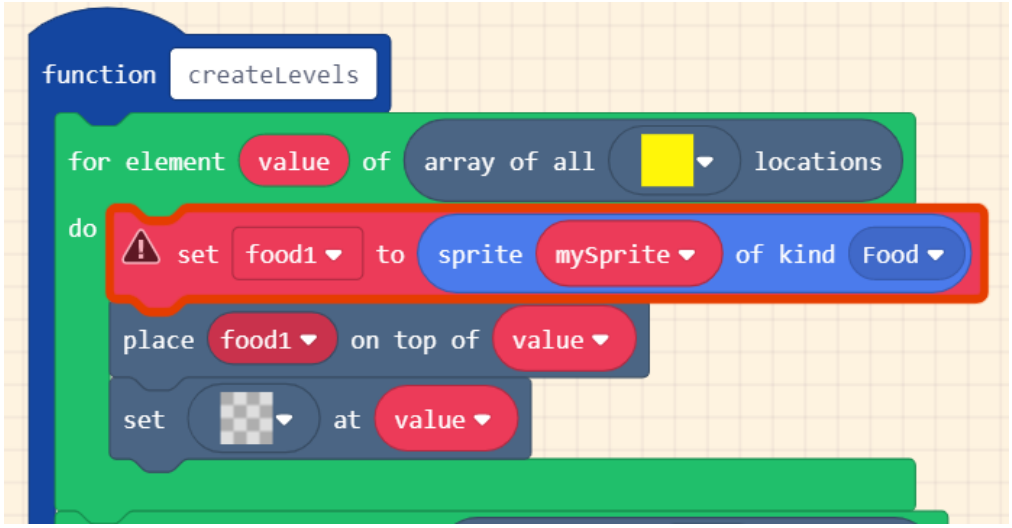
debugging

The problem may be to do with the order of your code. The code in the screenshot here tries to move a sprite to a location but before it has even been created. If you get one of these code try reading through your code to see if it is the right order.

A similar type of error is a **Syntax Error**. When there is a syntax error in MakeCode Arcade your game will not run and may just show a grey version of your game in the preview window.

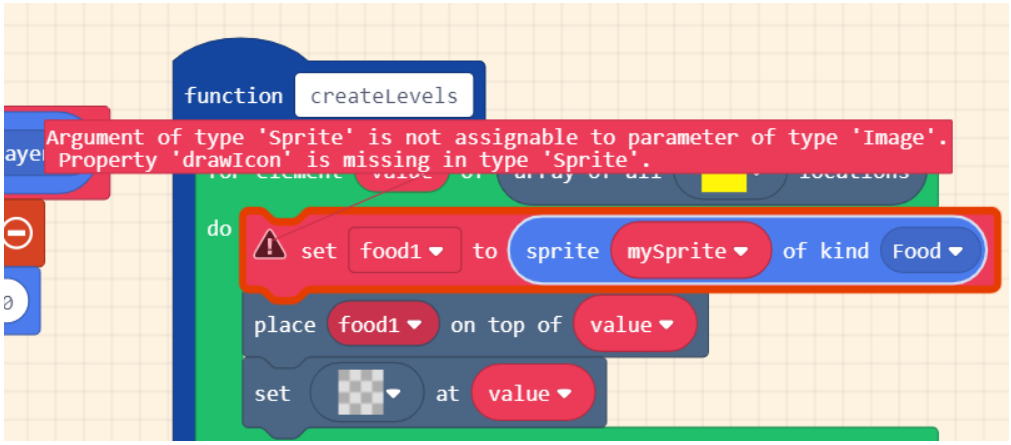
debugging

When this happens have a look for the **!** sign in a hazard triangle somewhere in your code. This is where the problem is.



debugging

Click on the triangle and you will get more details of what the error is about.



debugging

This may not make a lot of sense to you. But often it is due to putting the wrong kind of block in the gap. In this case there should be an Image block in this gap. The code error message can help us debug our code as even if we don't fully understand it, at least we know where the problem is. We can try out different blocks in this space to solve the problems. Also you can use the undo button at the bottom of the screen to go back in stages to a situation where the code was working. If you have made a few changes keep pressing that buttons until the game preview is in colour again.

Finding and Fixing Glitches and No Behaviour Bugs

This is not universally accepted as a difference but for the purposes of this course bugs and glitches are different.

- Bugs are mistakes in the code where the intended effect doesn't happen so you can't achieve what you want
- Glitches are mistakes where unintended effects do happen that are different from what you wanted

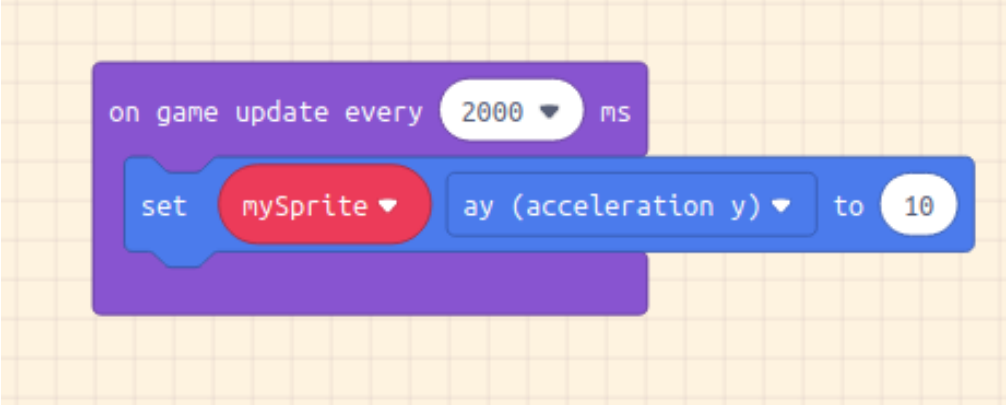
Bugs and Glitches often show up when we are testing our game. Sometimes they show themselves quickly when we are self-testing, sometimes it takes other people playing the game to find glitches that's why play testing is so important.

While bugs are often just frustrating, we can sometimes celebrate glitches. They can be fun and are normally a good way for us to understand more about what is happening on a code level.

One technique you can use to try to solve glitches or bugs is through debugging with a technique called **watching variables**. To do this click on the image of the bug under the game preview window.

This brings up a new window and also changes how the blocks look on the block building part of our screen. It puts little squares next to them. These allow us to set code break points. A break point is a way of stopping the running of the game code at a certain point. In our case let's use an example were we do that to check the value of a variable we have created.

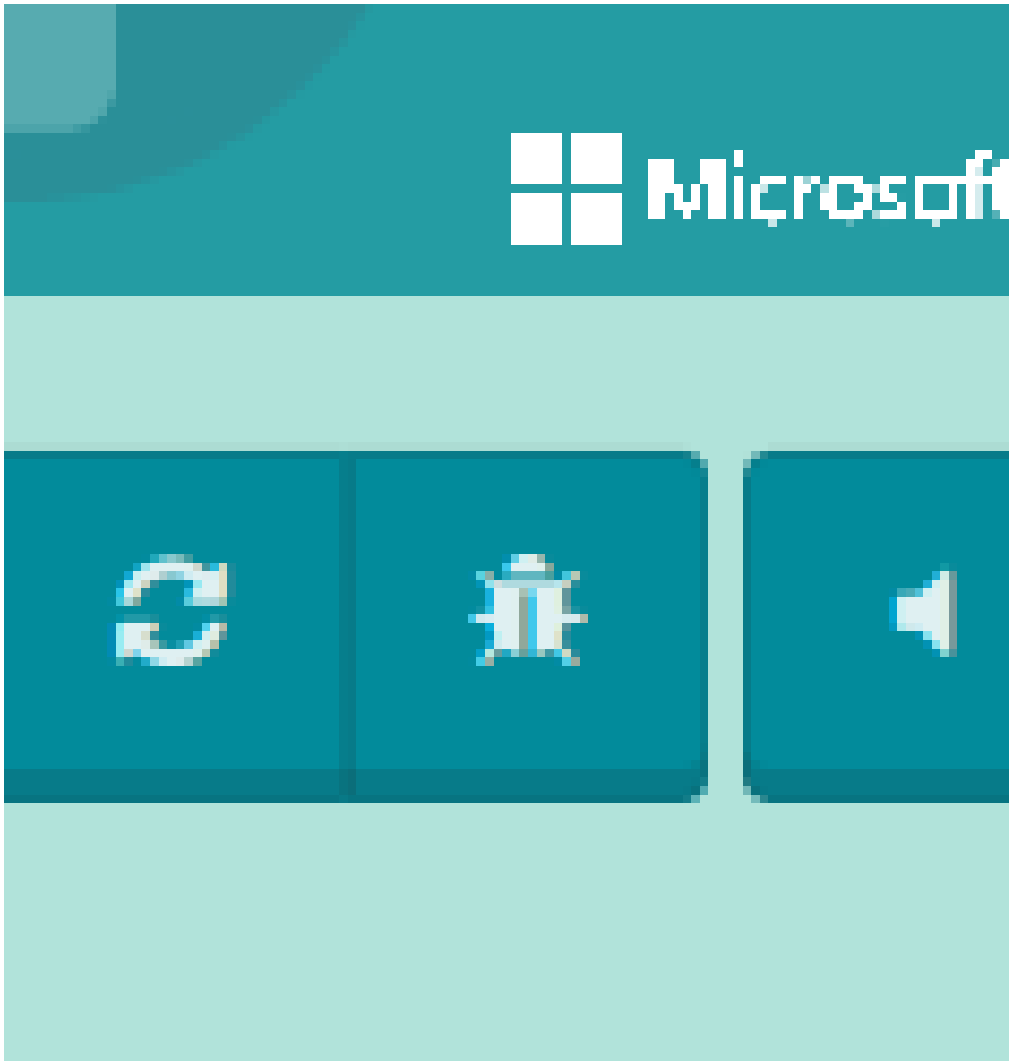
Imagine we want to increase the gravity every two seconds as the game progresses. Just a little, so the game gets harder as time goes on. The desired behaviour is to change the gravity so it increases by 10 each 2 seconds. To do this the following blocks are tried.



debugging

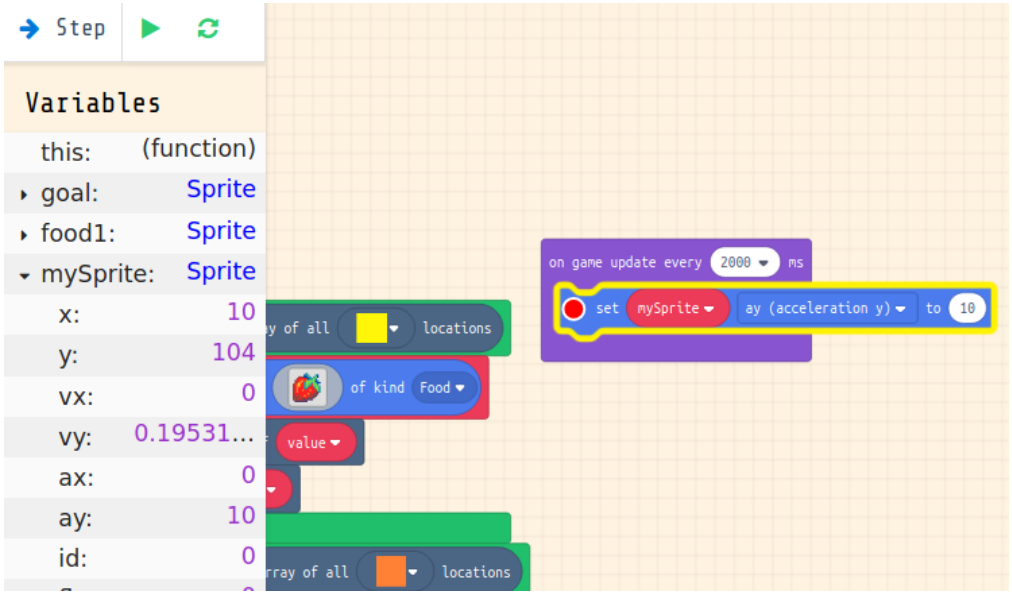
The desired behaviour doesn't happen. The player gets lighter not heavier. You may see the mistake right away but imagine you don't.

Turn on debugging by clicking on the image of the bug under your preview screen.



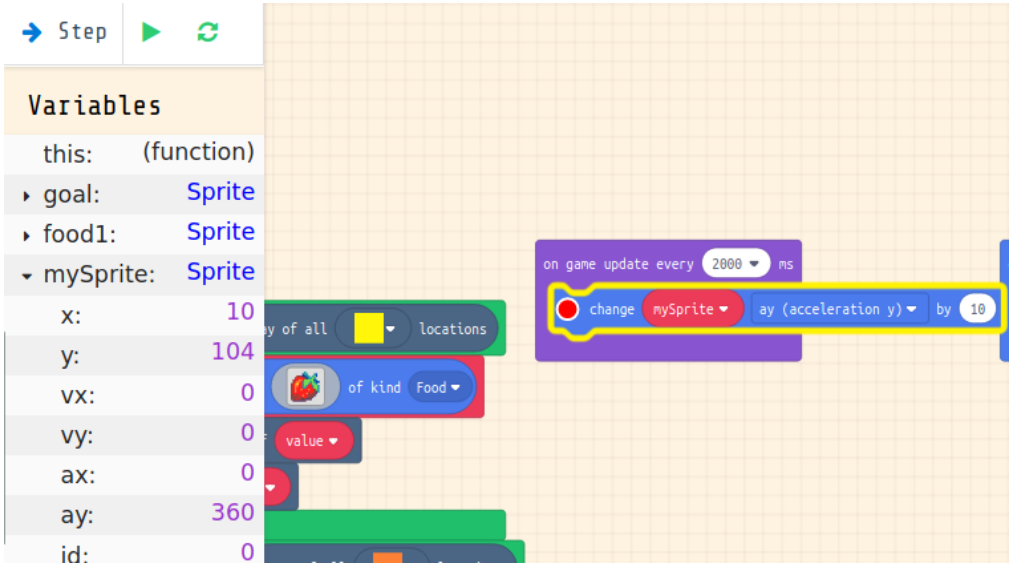
debugging

We are going to set break points in our code to see what is happening. Do this by clicking in the circle next to the code we are interested in. In this case in our game update loop.



debugging

Now click on the green Play button to start running the code. It will stop when it gets to the break point. We can now see what the value of the variable we are interested in is at this point. In this case click on the arrow next to mySprite to see what the ay (gravity) variable is. We can see that instead of increasing by 10 each 2 seconds it is being set directly to 10 instead.



debugging

Now we know this is the cause of the error we can change the block to **change mySprite ay by 10** instead. If we repeat

the debugging then we'll see the value go up in steps of 10 each time we click on the green play arrow to step through the code.

When you are self-testing or play testing games and find mistakes find out the following:

- What type of error are you encoutering - Errors, Bugs or Glitches?
- If they are black or grey screen errors can you find any messages?
- If they are bugs glitches what is the different between the intended behaviour or effect and the actual behaviour?
- What tactics are there to solve the errors, bugs or glitches?

5 Missions

The resources here are part of a learning model which has the following elements:

- Game Making MISSIONS main missions involve choosing authentic game making design patterns from a menu. Other side missions help with engagement of different making preferences.
- A MAP of Learning Outcomes particularly suited to digital game making, presented in an accessible format to teachers and learners.

- Design METHODS are techniques that facilitators and learners can use to help navigate the process of making a game and reflecting on what is being learned in the process.

This page outlines different kinds of missions that are used in this game making approach. It needs to be rewritten to make it concise and with nods to existing, similar approaches.

Methods Using Missions

- Macro Mission - Tell a Systems Story (environmental or social) through a game

- Improving Half-Baked Game
- Player and Maker types & Specialised missions
- Limits to the Mission based approach - meeting your self in the middle

5.2 Using Game Design Patterns as Missions

One of the driving ideas of this approach is to choose game making design patterns from a menu

5.3 Main Macro Mission

The starting hook for our mission is to make a game which tells an environmental or social story. This approach is sometimes called Games for Change. The game doesn't need to be too serious

and of course it can be a lot of fun. Having a bit of a guide for what kind of game to create can be helpful if learners are unsure of where to get started.

5.4 Navigating with Game Design Patterns

The key idea of this approach to game making is to start with a simple Platform Game Template to Remix and to add Game Patterns from a menu of possibilities.

mechanics space polish and systems

5.4.1 Improving Half Baked Games

This process is a way to jump right into altering code. Instead of planning from first principles, learners are given a game that is designed to provoke players to tweak it. This process is inspired by game modding and a process called using half-baked games .

We start coding by jumping right in. Try to play this game. It's broken right? You can't jump high enough. Now look at the code of this broken game and use the link to workshop cards with activities to fix the game and add some elements quickly.

- Broken Platform Game to Fix
- Supporting Activity Cards

5.4.2 Extra Missions (suits Socializers and Griefers especially)

Extra Mission Cards These extra missions encourage us to think of our games as dynamic systems that change as we play the game.

Mission Cards for this Sessions

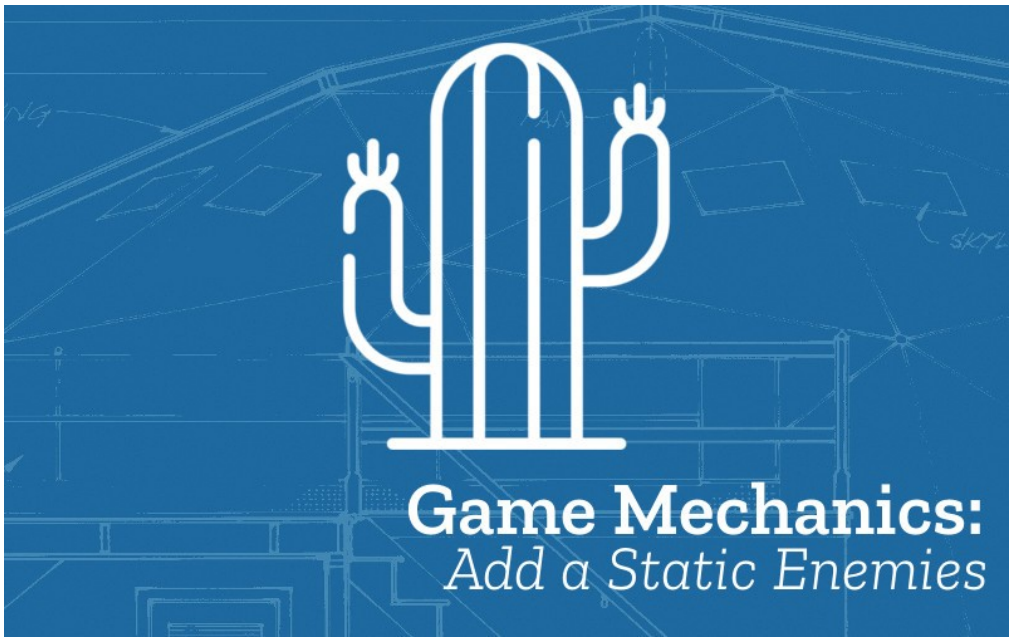
You can point them towards elements of systems thinking that arise from these challenges. These may include the following.

- Systems Elements
- Systems Dynamics
- Balancing Feedback Loops
- Reinforcing Feedback Loops

Social Mission Cards Public Missions

Secret Mission Cards Secret Missions

6 Add a Static Enemy



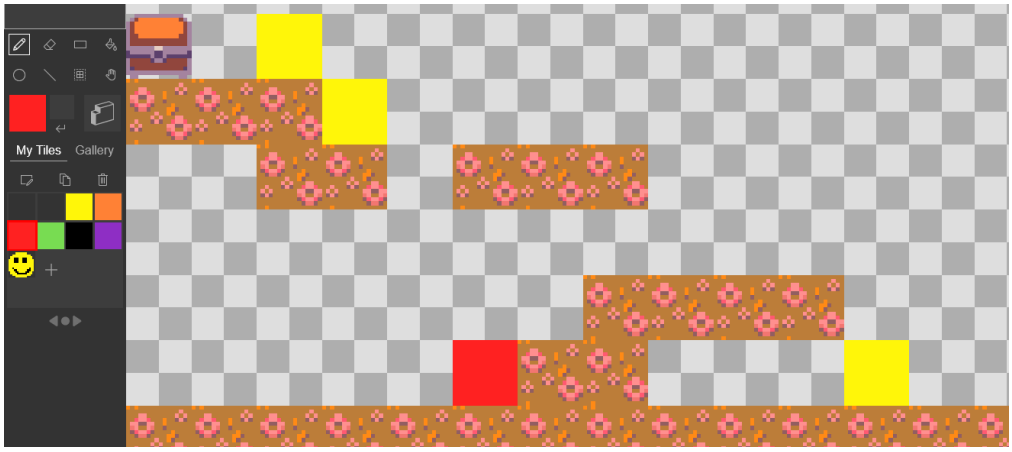
mechanics add a static enemy

- **Name:** Add a Static Enemy
- **Description:** Also known as a Hazard, a Static Enemy will kill or damage the health of the player if they touch it. These are often placed in tricky spots which the player is likely to bump into when jumping or trying to collect rewards.
- **Need for Pattern:** Having hazards increases the challenge of a level, you can place hazards in a way that requires the player to time their jumps well and really control their movement.
- **Related Game Patterns:** Add Moving Enemies [related], Jump on Enemies [related]
- **Coding Concepts involved:** Loops, Events
- **Links to other Computing Patterns:** , Change Listener, Input Event

6.1 How to implement this Pattern in MakeCode

6.1.1 Add Static Enemies to your Tilemap

Click on your tilemap. Create a totally Red tile in **My Tiles**. Add one or two red blocks to your first level.

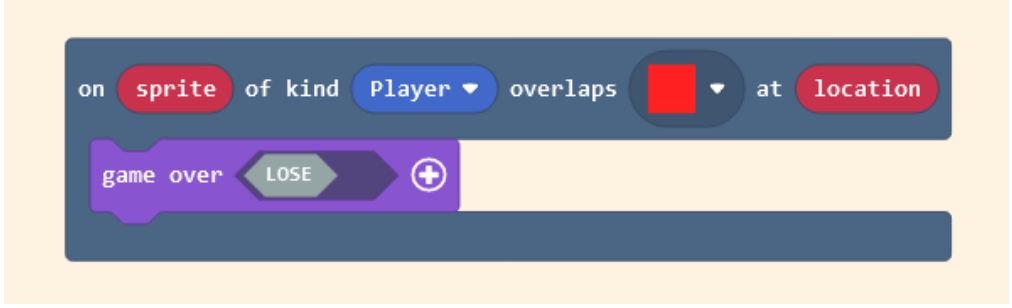


Static Enemy Block

You can also draw an enemy / hazard of your choice in My Tiles section. Click on the plus sign to do that.

6.1.2 Create a Collision Listener

We now code what happens when our player overlaps with our **staticEnemy**. Drag in an **on sprite of kind player overlap with __ at location** from Scene. Inside the block drag in from Game block of **game over** and keep it set to **Lose**.



Static Enemy Block

6.2 Test your game and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. In this case check that each time you add in a red block in your level tilemap/s it should behave as a static hazard. When you touch the enemy the game ends with a Game Over message.

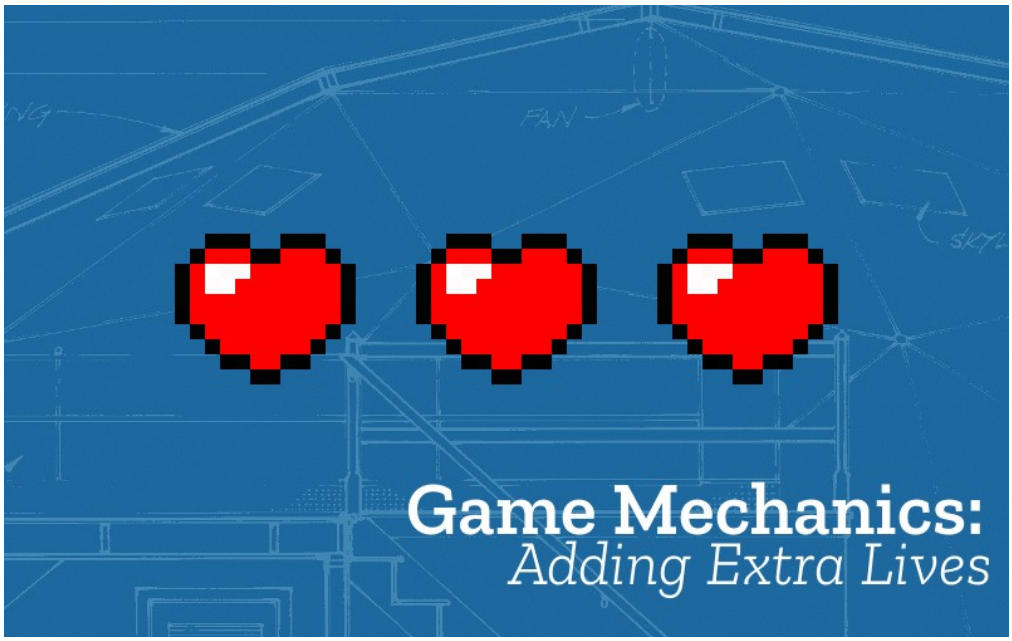
To check that you are making the most of this pattern you can ask yourself the following questions:

- Do you have any enemies in places that make it tricky for your player when they jump.

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

As a next step you may find you want to increase the challenge even more perhaps by adding moving enemies. Or you may find that you want to balance out the increase of challenge that these hazards have brought and add player lives.

7 Add Player Lives



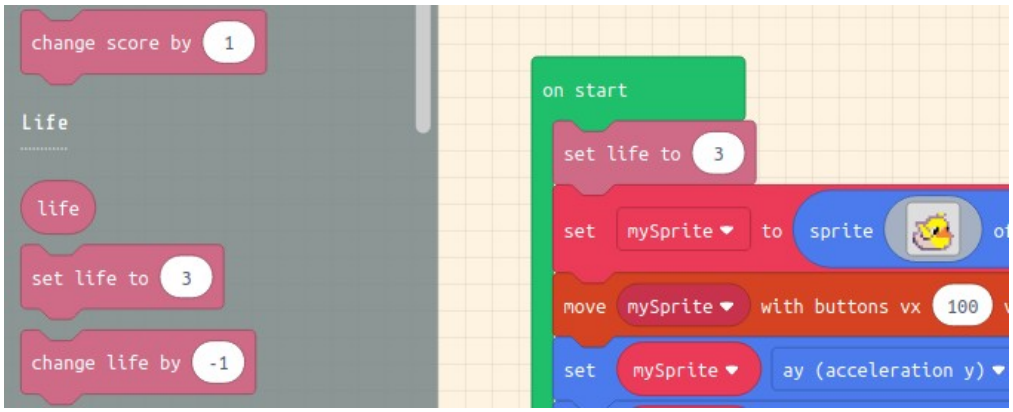
mechanics space polish and systems

- **Name:** Add Player Lives
- **Description:** The player starts the game with a number of the lives When the Player is zapped, or runs out of health the play restarts but with one less life.
- **Need for Pattern:** Having player lives is a way of reducing the frustration of a challenging game. For example players normally restart from the level they got lost their life on rather than going back to the very beginning.
- **Related Game Patterns:** Before adding this pattern you'll need something that can zap you. So add a pattern like **add Static Enemy** or a moving enemy.
- **Coding Concepts involved:** Variables
- **Links to other Computing Patterns:** Systems Dynamics,Change Listener

7.1.1 Adding a starting amount of lives

We can add in the starting number of lives. To do this drag in from **Info** a **set lives to 3** block to

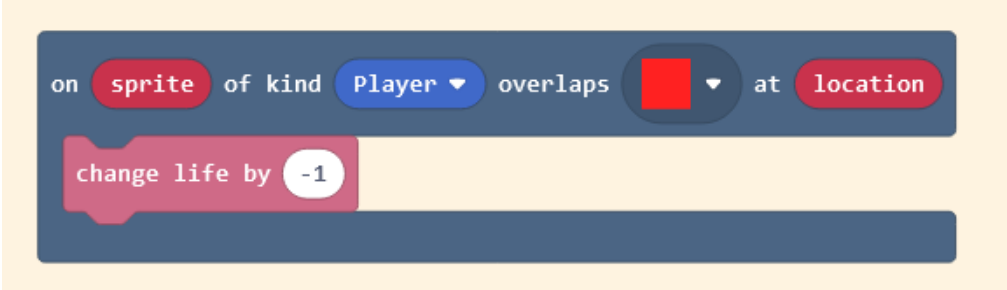
the beginning of the on start loop.



mechanics space polish and systems

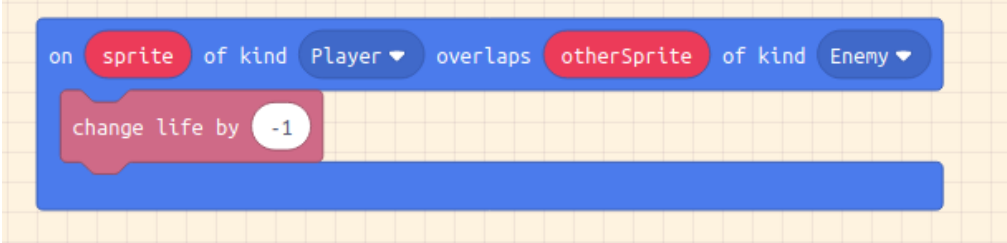
7.1.2 Change the overlap loop for our Enemy

In this game previously you may have had code where the player overlapping with a Static Enemy would be Game Over with the player losing.



lose a point

Another possibility is that you may have moving enemies with a different code structure. You can change to replace the Game Over lose block with a **change lives by -1** block.



lose a point

7.2 Reflecting on what's happening

Variables: In this example you are using a Variable of called life to keep track of the player lives. In this example you start with three and you update the value of that Variable everytime you bump into an enemy.

Change Listener: The **on sprite kind overlap** block is always listening out for a change where if there is a new overlap with an enemy the programme will run the code inside that block. This kind of Change Listener is often used in computer programmes to react to new situations.

Systems Dynamics: The use of new lives means that the balance of the game will be altered. This may make it easier so you may need to balance this out by making something in your game harder. This process of balancing out systems elements allow us to explore Systems Dynamics.

7.3 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example check that each time you touch an enemy your number of lives goes down by one.

There is a known side effect with some kinds of enemies where you lose more than one life. To avoid this you may need to **make Player Immune** pattern

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

8 Add a timer

mechanics space polish and systems

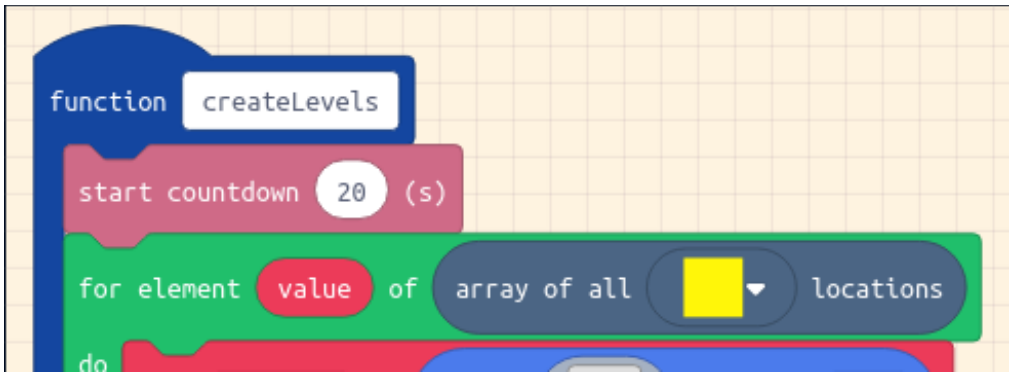
- **Name:** Add a Timer
- **Description:** The player is required to complete the level or another goal before the timer runs out.
- **Need for Pattern:** Having a **timer** is a way of creating challenge for the player. If you have more than one level you can reduce the time allowed for the next level to increase the challenge as the game progresses.
- **Related Game Patterns:** Add Player Lives [related], Add More Levels [related],
- **Coding Concepts involved:** Data
- **Links to other Computing Patterns:** Systems Dynamics, Making Functions

8.1 How to implement this Pattern in MakeCode

8.1.1 Simple Timer for each level

Drag in from the **Info** section a block that reads countdown into the createLevels function.

Because this is in the createLvels function rather than in the on start function, this timer will reset everytime you reach a new level.

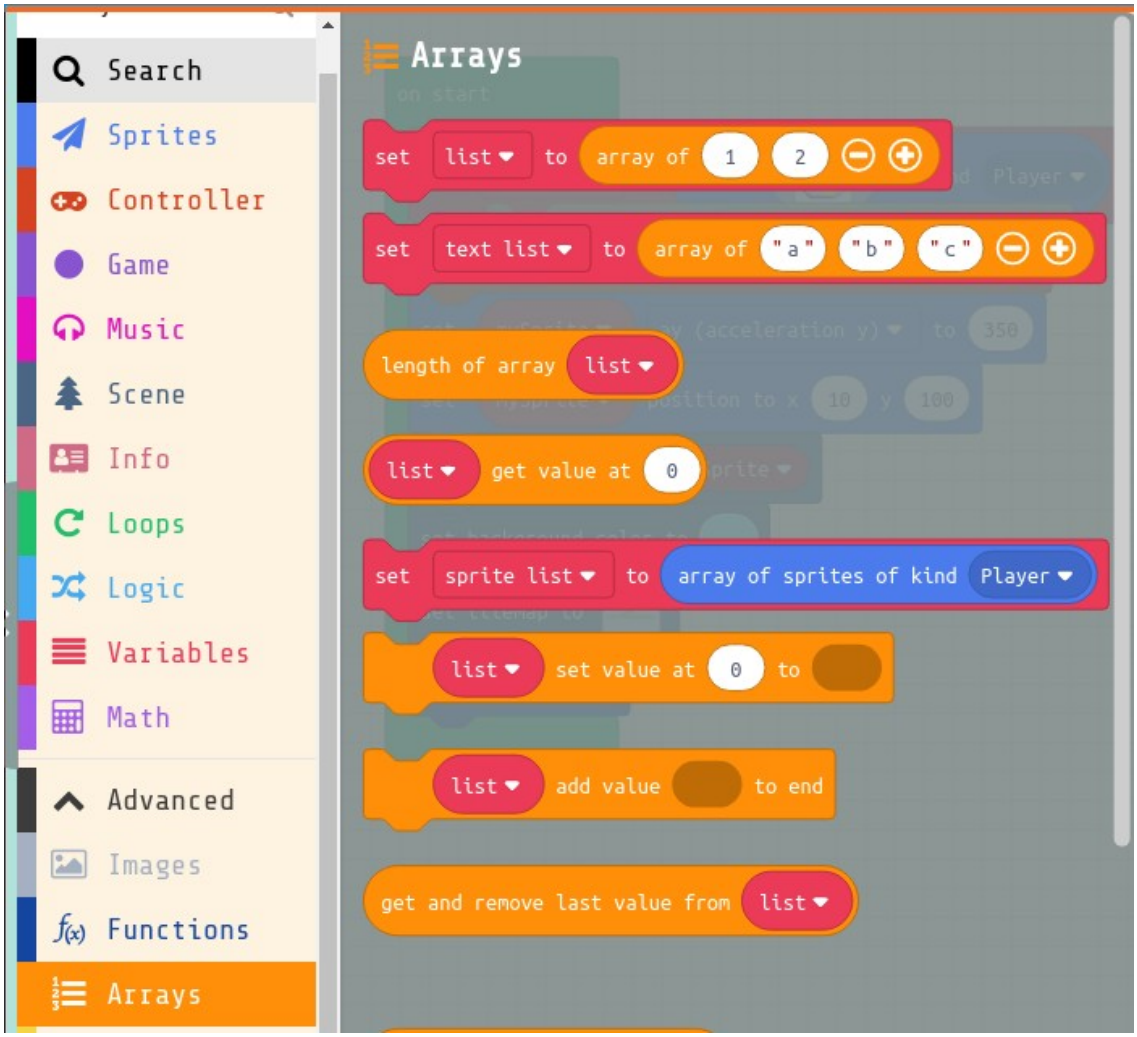


add timer

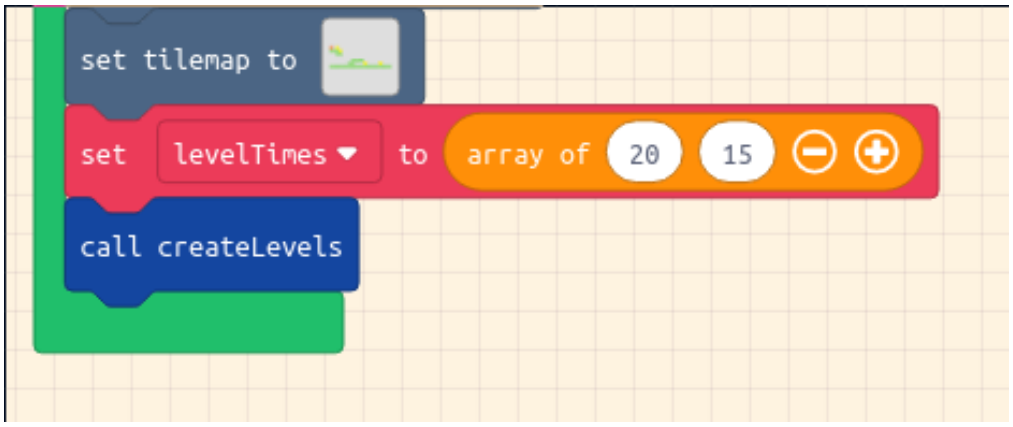
8.1.2 Have different timer values for each level

One classic way of adding challenge to a game is to ask your player to solve the next level of your game in less time. If you want to have different values for your timer for each level. We will create an array variable which will contain a list of the different times.

From **Advanced > Arrays** drag in a **set list to array of** block to the end of the on start block above the **call createLevels** block.

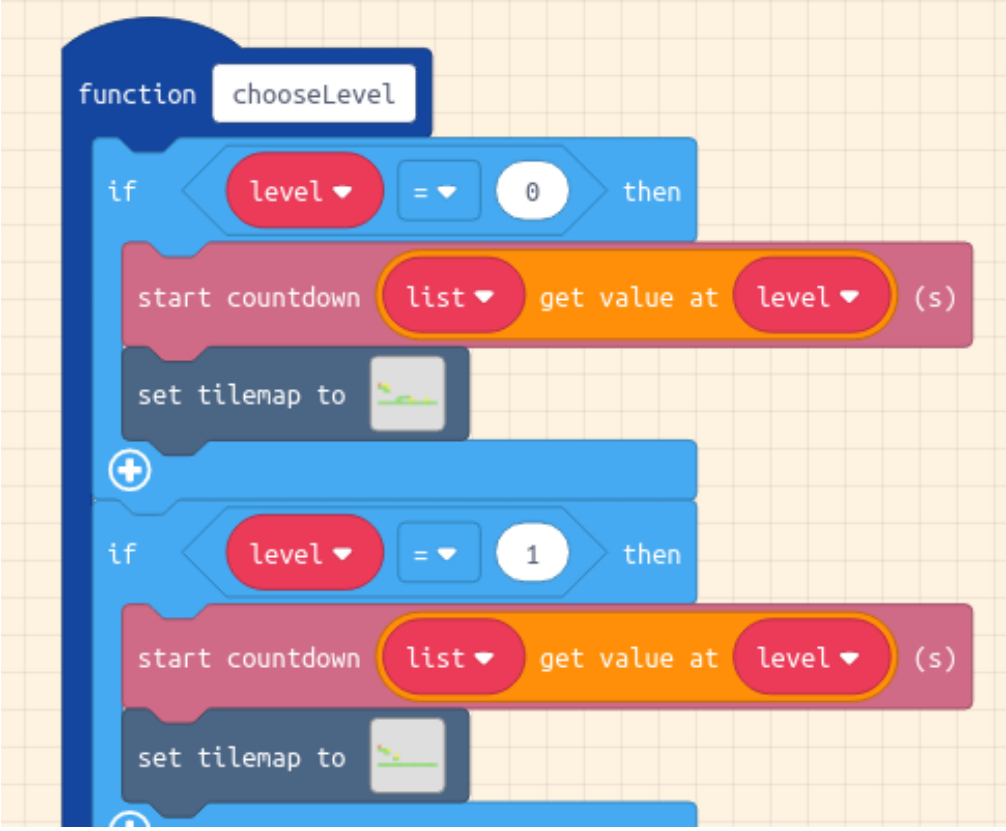


Then change the variable list to one called **levelTimes** by clicking on it and selecting New Variable. Then change the times for each level. I'm setting the first level time to be 20 and the next one to be 15.



If you haven't already drag in from the **Info** section a block that reads countdown into the createLevels function for each level. Next from **Advanced > Arrays** drag in a **list get value at**

block and add it to the countdown block, and include your levelTimes value at level, as you can see in the screenshot below.



8.2 Test your Changes and Next Steps

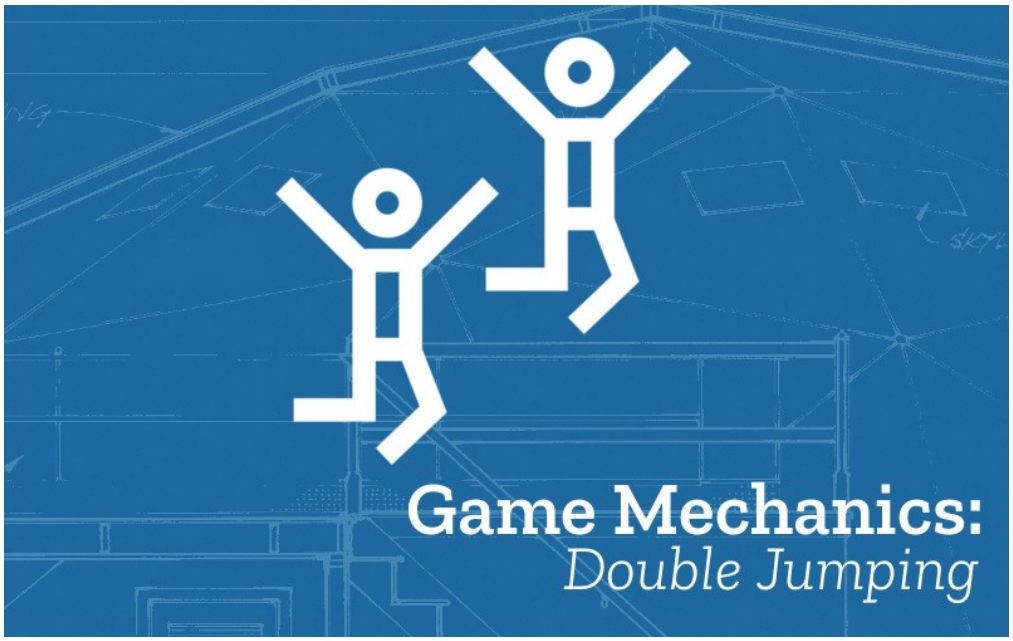
Test your game to check that your changes have the desired behaviour and that there are no side effects. To check that you are making the most of this pattern you can ask yourself the following question/s:

- Make sure you haven't made your game too hard or too easy get others to test it

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

Some next steps you might want to add may be add more Levels. If you do that then you will need to look at the pattern to add different timings for different levels.

9 Double Jump



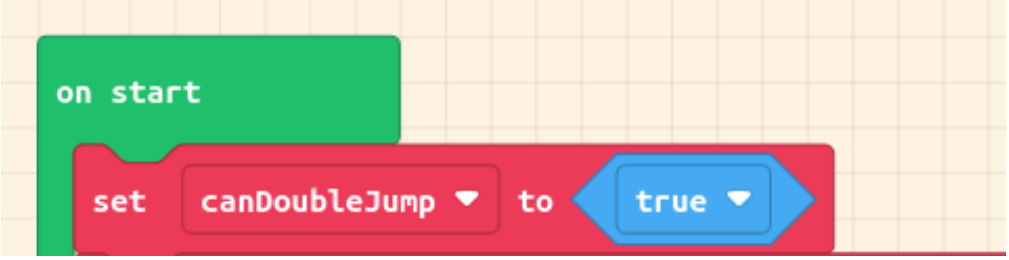
mechanics space polish and systems

- **Name:** Double Jump
- **Description:** The player is able to jump in the middle of another jump but only once. Pressing the jump button a third time with have no effect.
- **Need for Pattern:** Having a double jump is a way of allowing the player to access areas not possible with a single jump. Double Jumps also require good timing from the player too so this can increase challenge depending on the design of your game.
- **Related Game Patterns:** Jumping on Enemies [related]
- **Coding Concepts involved:** Data, Change Listener
- **Links to other Computing Patterns:** Systems Dynamics,

9.1 How to implement this Pattern in MakeCode

9.1.1 Create a “canDouble Jump” variable

We need to create a variable called canDoubleJump. Now add a block at the start of our game and set it to true to start with.

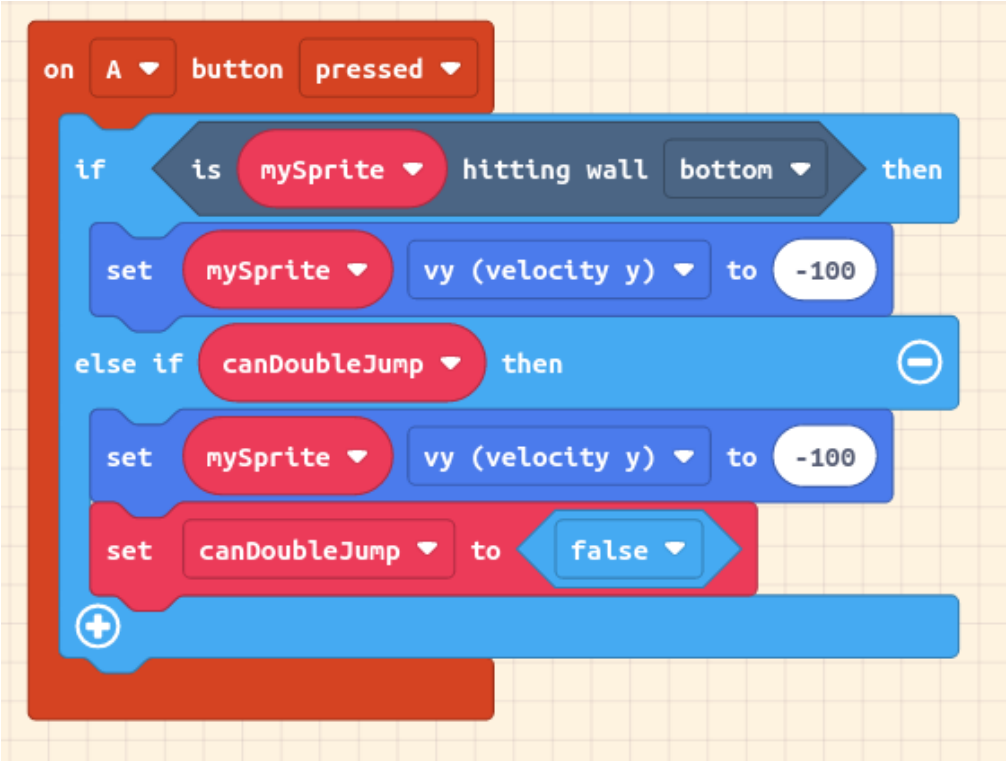


Double Jump 2

9.1.2 Create a Logic block to test if player can jump or not

Then you need to use blue logic blocks to check to only jump if

- the bottom of your player (mySprite) is touching the floor (wall) or
- or if (else if) canDoubleJump is true



Double Jump 3

Once the player has used up their double jump ability, you set the canDoubleJump variable to false. This stops the player from being able to jump more than twice.

9.1.3 Reset canDoubleJump variable

Then we must create to code to reset the canDoubleJump variable when you touch the ground again. To do this create a on game update loop and put a logic block in there which will turn canDoubleJump true when you are touching the ground again.

Double Jump 1

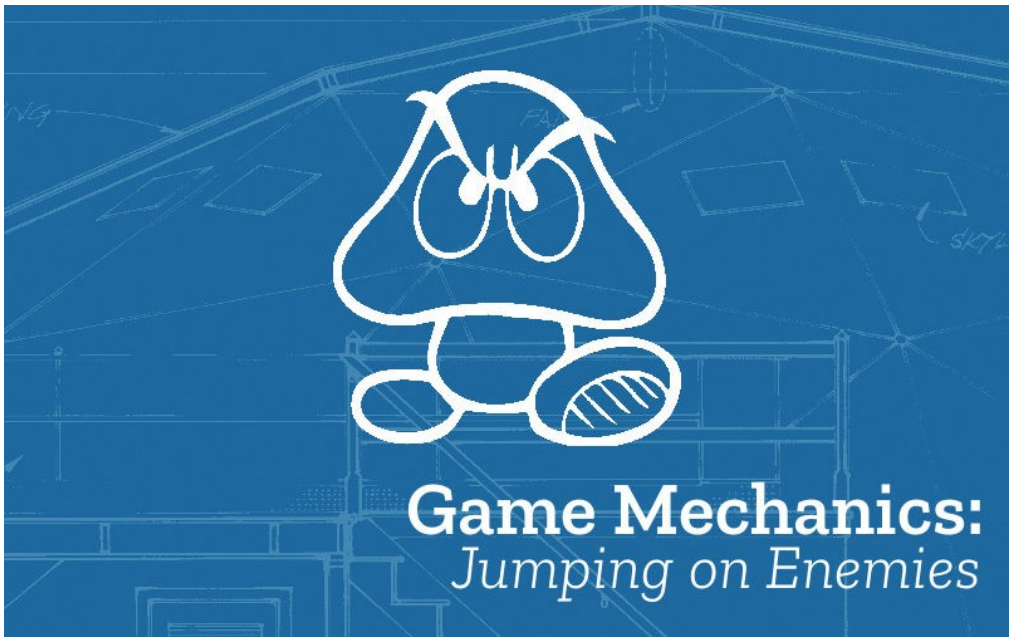
9.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and no side effects. To check that you are making the most of this pattern you can ask yourself the following questions:

- Now you have a double jump should you reduce the normal jump height (via changing the jump velocity or gravity effecting the player)
- Are there any parts of your game that you can only access via a double jump

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

10 Jumping on Enemies to Zap them



Jumping on Enemies to Zap them

- **Name:** Jumping on Enemies to Zap them
- **Description:** In many games players shoot enemies. In some platformers they get rid of them by jumping on them instead. If the player is descending from a jump when they touch the enemy the player is zapped and in this case disappears.
- **Need for Pattern:** Being able to jump on an enemy is a good way of clearing the area you want to explore. You may need to have a clear space to be able to jump up to a high platform for example. Some platformers do have a shooting mechanic as well but using this pattern and sticking with jumping also keeps the game simple (in a good way).
- **Coding Concepts involved:** Data, Change Listener
- **Links to other Computing Patterns:** Systems Dynamics,
- **Related Game Patterns:** You'll need to have added the **Add Enemies** pattern to your game before you can add this one.

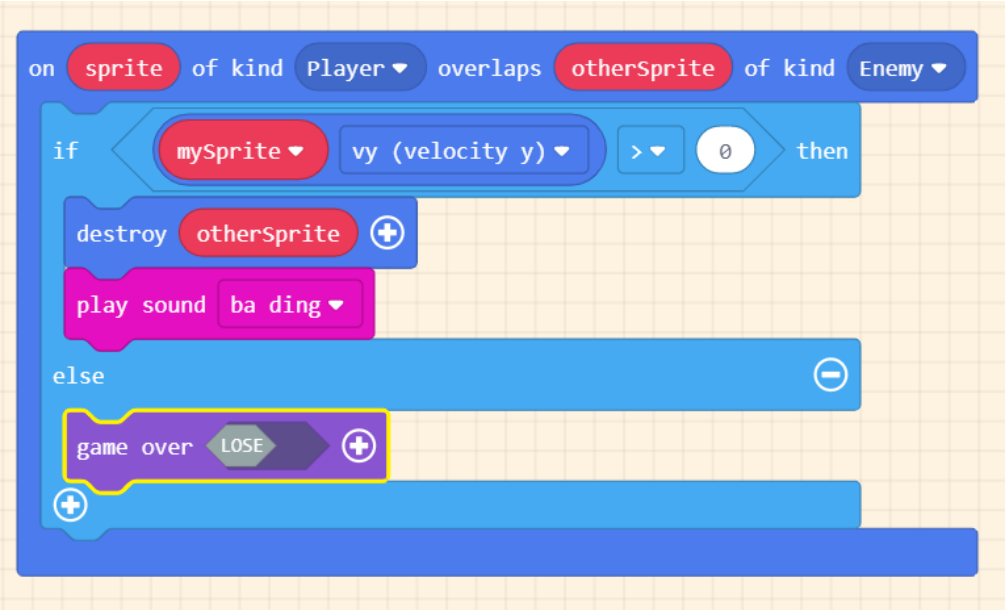
10.1 How to implement this Pattern in MakeCode

10.1.1 Add a Condition for the Overlap Listener block

We want the enemy to zap the player if they overlap normally or if the player is jumping up. But we want the player to zap the enemy if they are travelling on the down part of their jump.

To do this we will check the Player's x axis velocity. As the numbers are measured from the top of the screen going down, if it is greater than 0 then the player is travelling in the down direction.

Check if that's true using the code blocks below, and if that isn't the case then set the game to game over.



Jumping on Enemies to Zap them

The blocks in the **else** section may be different for example if you are using player lives, you will want your player to lose a life instead.

10.2 Test your Changes and Next Steps

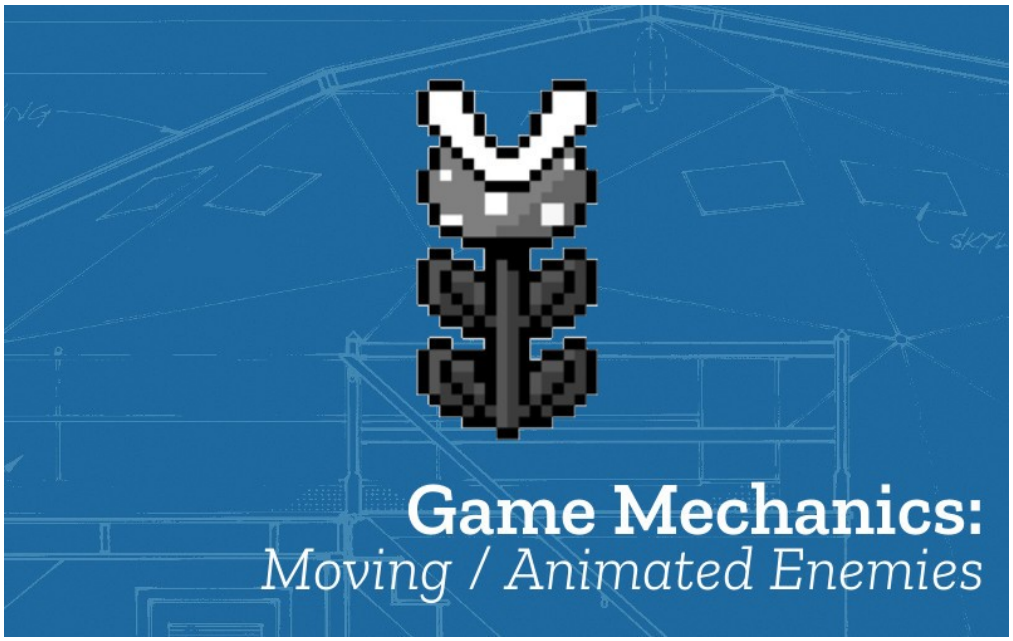
Test your game to check that your changes have the desired behaviour and that there are no side effects.

To check that you are making the most of this pattern you can ask yourself the following questions::

- When you add this mechanic you may be making the game much easier. Is there anything else you can do to make it more challenging?
- Can you imagine the player experience? Is there anything you are fogetting or taking for granted?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

11 Moving Enemies - Animated



moving enemies image

- **Name:** Moving Enemies Animated
- **Description:** In this pattern a static enemy is animated to make some limited movements around its home position, for example bobbing or bouncing.
- **Need for Pattern:** Having a moving enemy is a way to increase the challenge of the player. It also gives a sense of movement and excitement to the game.
- **Related Game Patterns:** Add Static Enemy [required], Add Moving Enemies Patrolling [related]
- **Coding Concepts involved:** Data, Events, Loops
- **Links to other Computing Patterns:** , Change Listener, Systems Dynamics

11.1 How to implement this Pattern in MakeCode

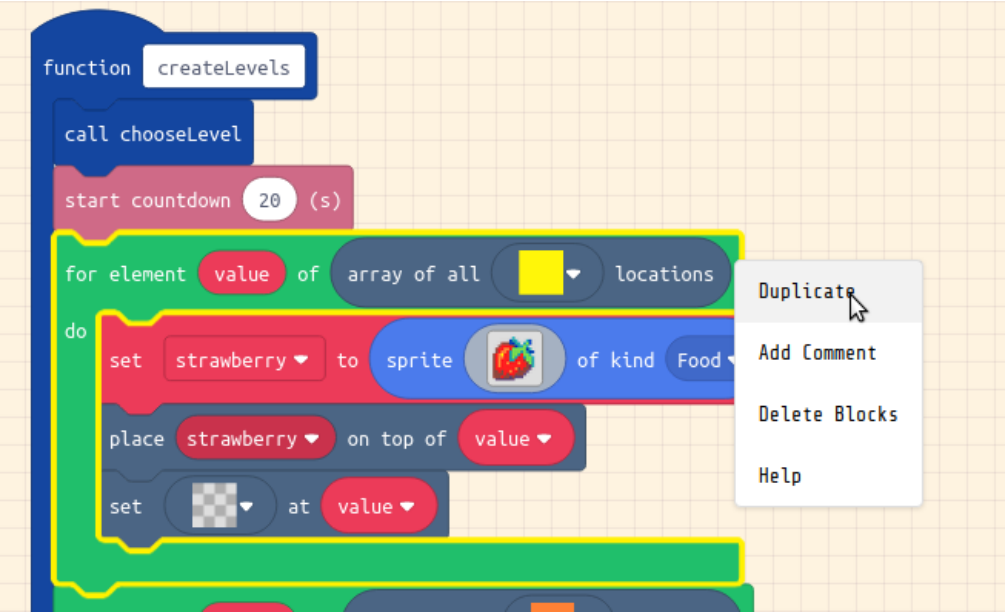
11.1.1 We add enemies like we add food.

We add enemies like we add food to the game. Following this tutorial will add static enemies to your game. Click on the tilemap image for your first level. Create a totally Red tile in **My Tiles**. Add one or two red blocks to your first level.

Add Red Blocks

11.1.2 Looping through the tilemap squares

For first line here reads **for element value of array of all...** This line contains a value and a list. The loop keeps running until it runs out of a values in the list. In this case create one item of Food for every yellow block. ### Duplicate the Food Loop Duplicate this loop section.

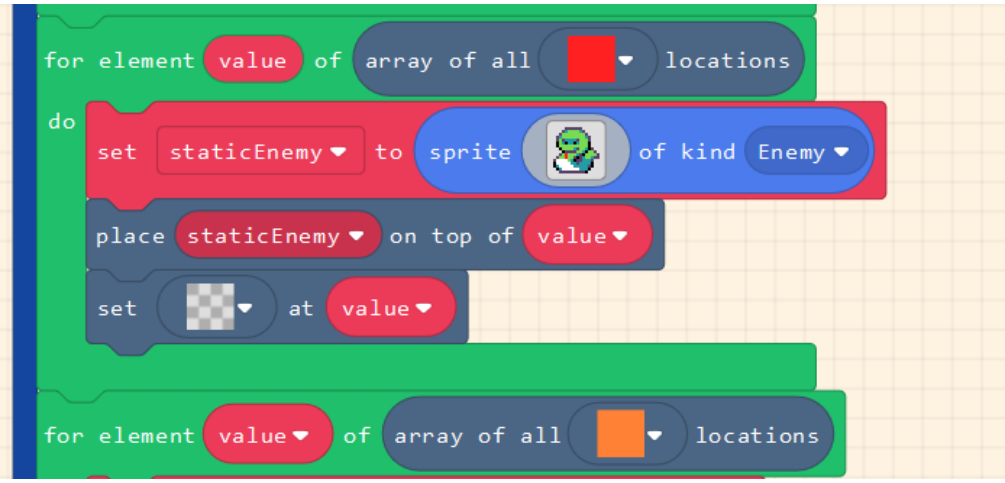


Now we will change the **for loop** will turn the red squares in a tile map into a sprite of kind of Enemy.

11.1.3 Change the values for our loop to create enemies

Drop the copied loop back into the function after the original one.Change the values of content of this **for loop**. Change the yellow square to a red one. Change the name of **food1** for the two blocks for it to mentioned. I'll choose to create a new variable called **staticEnemy** and change the image too this time to snake.

Check your code with the example below.



11.1.4 Create a Collision Listener

We now code what happens when our player overlaps with the enemy our **staticEnemy**. Drag in

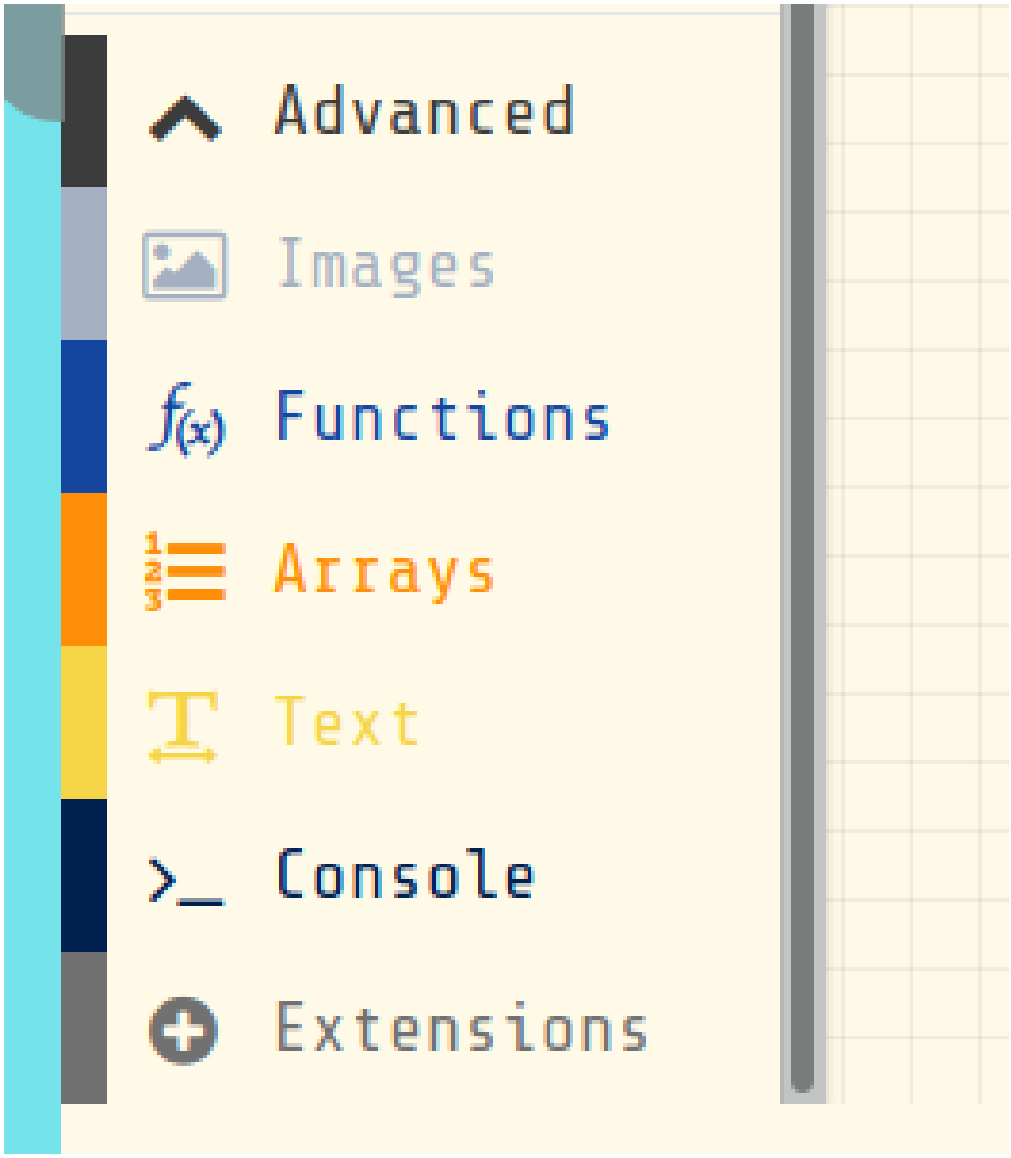
an on player overlap with block from Sprites. Set the second value to be **Enemy**. Inside the block drag in from Game block of **game over** and keep it set to **Lose**.

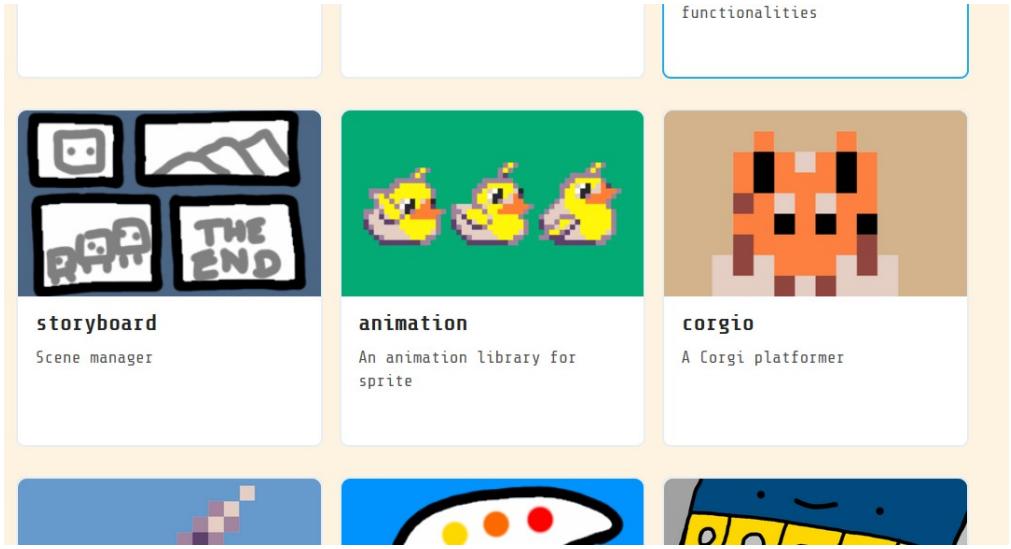
Static Enemy Block

11.1.5 Animate our Enemy (Optional)

We can add a bit more challenge to our game by making otherwise static enemies move around a fixed point using animation.

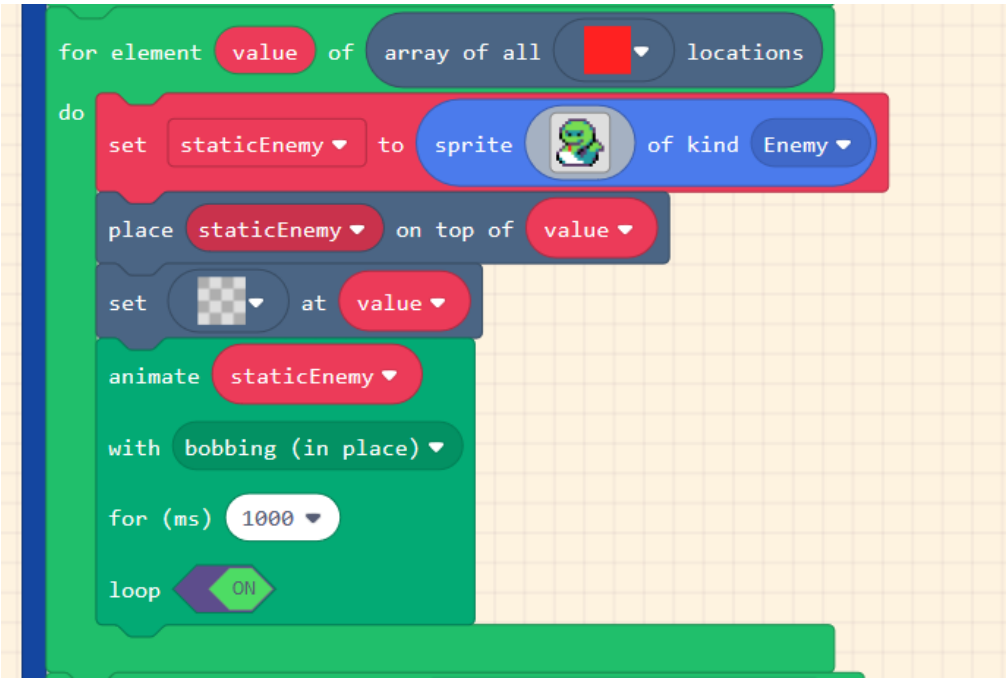
We alter our static enemies to animate them. To allow us to do this go to **Advanced > + Extensions > Add Animation**





We can now drag in an animate block which has an animation effect on it. Alter it so it matches the one below in our **for each element** loop so that it applies to all Enemies. I change the type of

animate from the drop down menu to use the simple bobbing in place animation.



animated enemies image

You can see here that for simplicity's sake I've have not changed the variable name of staticEnemy. But you can change it or create a new variable if you want to.

11.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example, one side effect that you might have if you already have more than one level is that enemies from a previous level may appear on your next level. See the Add Levels tutorial to fix that.

To check that you are making the most of this pattern you can ask yourself the following questions:

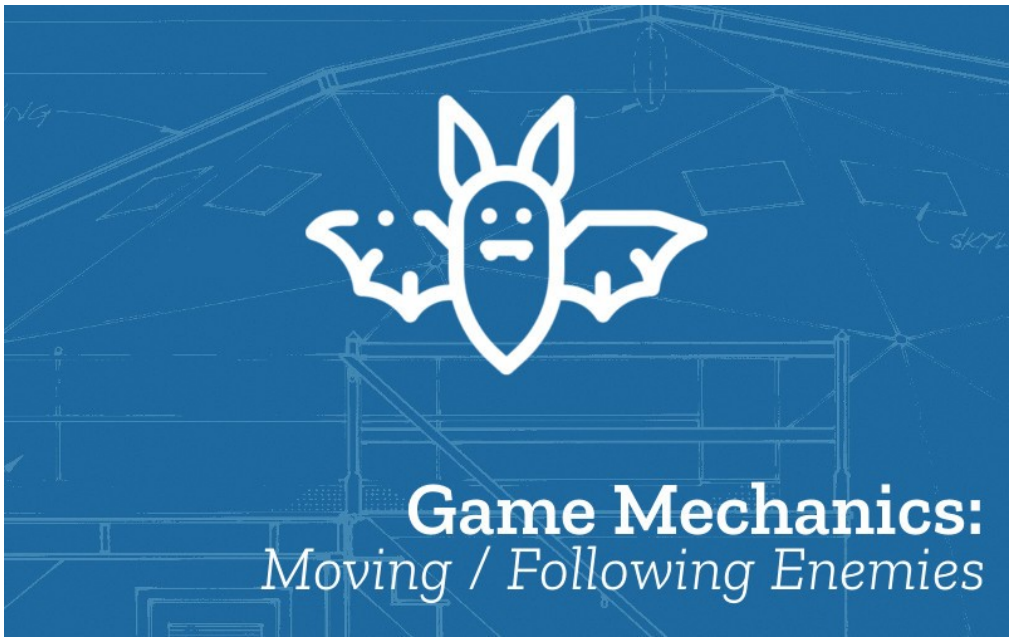
- Are there other types of animations that suit your game.
- Where can you place the animated enemies to maximise the challenge to your game.

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

In this pattern we replace our static enemies with animated enemies. If you want to have both kinds of enemies then you can do this by following the patterns shown in the **add a patrolling enemy** tutorial.

Also this pattern may make your game much more challenging. To balance it out a potential next step may be to add the **jump on enemies** pattern to your game if you haven't already.

12 Moving Enemies - Following



moving enemies image

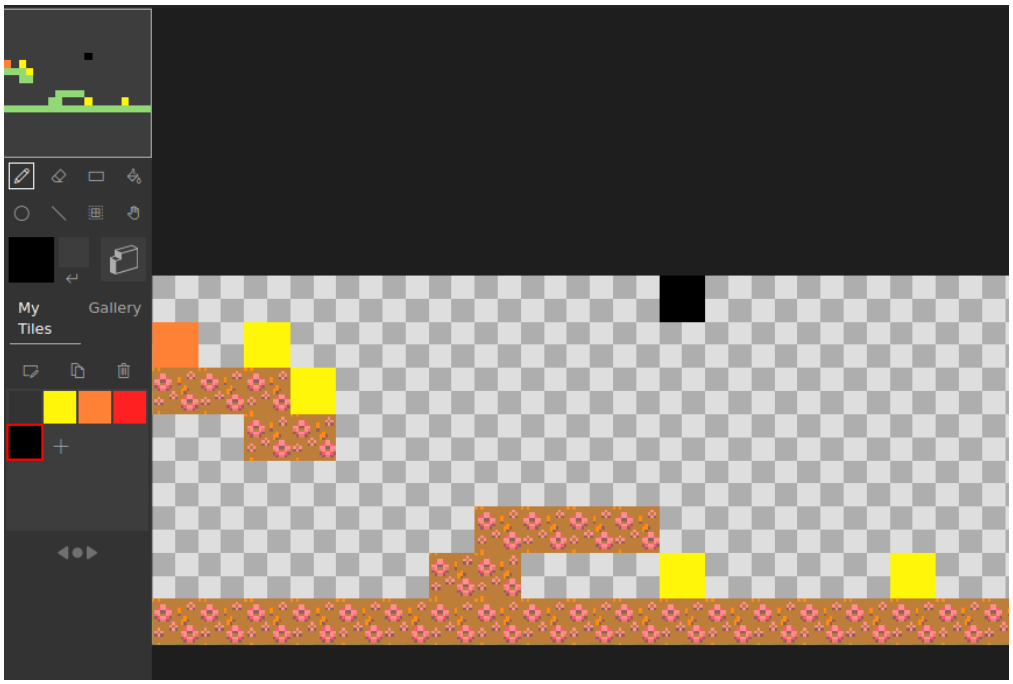
- **Name:** Moving Enemies Animated
- **Description:** Here one or more enemies try to chase the player by following them.
- **Need for Pattern:** Having a **following enemy** is a way to increase the challenge of the player to reach goals and to collect food. It also gives a sense of movement and excitement to the game.
- **Related Game Patterns:** Add Animatd Enemy [required], Add Moving Enemies Patrolling [related]
- **Coding Concepts involved:** Data, Events, Loops
- **Links to other Computing Patterns:** , Change Listener, Systems Dynamics

12.1 How to implement this Pattern in MakeCode

12.1.1 Step by Step instructions

This game mechanic works well when you have enough time to run away from these enemies or somehow get rid of them by shooting or jumping on them. This tutorial assumes you have already added a static enemy pattern.

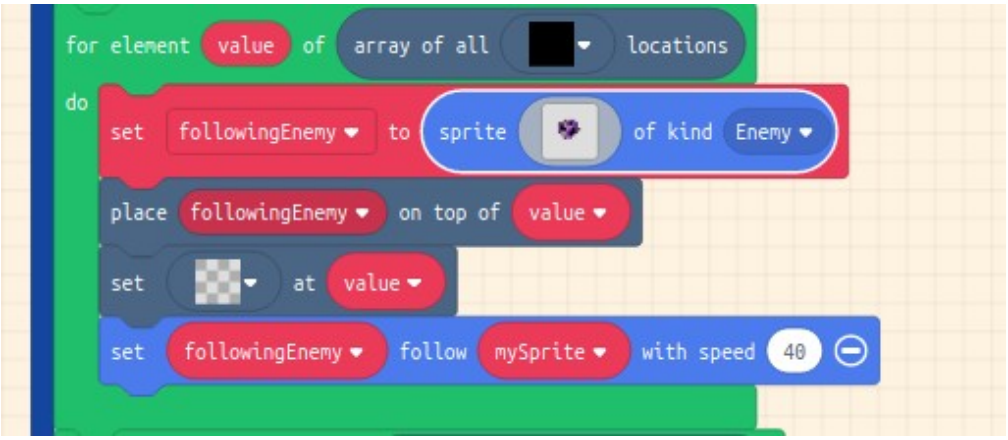
As with static enemies, we need to edit out tilemap and add a new colour of tile to our tilemap and place one in a location. In this case be sure to change your design to add some black blocks.



add another tile

Now follow the same kind of pattern for creating a static enemy but change the final blocks. I've chosen a small meteorite from the Gallery here. Having a small follower looks good.

Add a **set to follow** block from our Sprite section a **set followingEnemy to follow mySprite with speed 40**. Set this been to follow mySprite which is your player sprite.



add another tile

In the block above I've changed the speed of the bee to be slower at 40 than the default 100 as in this example the player just has to run away from the meteorite.

12.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

To check that you are making the most of this pattern you can ask yourself the following questions:

- Does this pattern make your game too hard? If so do you need to slow down the followers or reduce how many there are?
- Are there enough obstacles to dodge around to slow the followers down?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

This pattern may make your game much more challenging. To balance it out a potential next step may be to add the **jump on enemies** pattern to your game if you haven't already.

13 Moving Enemies - Patrolling

moving enemies image

- **Name:** Patrolling Enemies
- **Description:** In this pattern the enemy moves around in a particular area in a repeating pattern. This movement is called patrolling. The enemy does not try to seek out the player.
- **Need for Pattern:** Having a **pattern name** is a way to increase the challenge of the player to reach goals and to collect food. It also gives a sense of movement and excitement to the game.
- **Related Game Patterns:** Add Static Enemy [required], Jumping on Enemies [related]
- **Coding Concepts involved:** Data, Events
- **Links to other Computing Patterns:** , Change Listener, Systems Dynamics

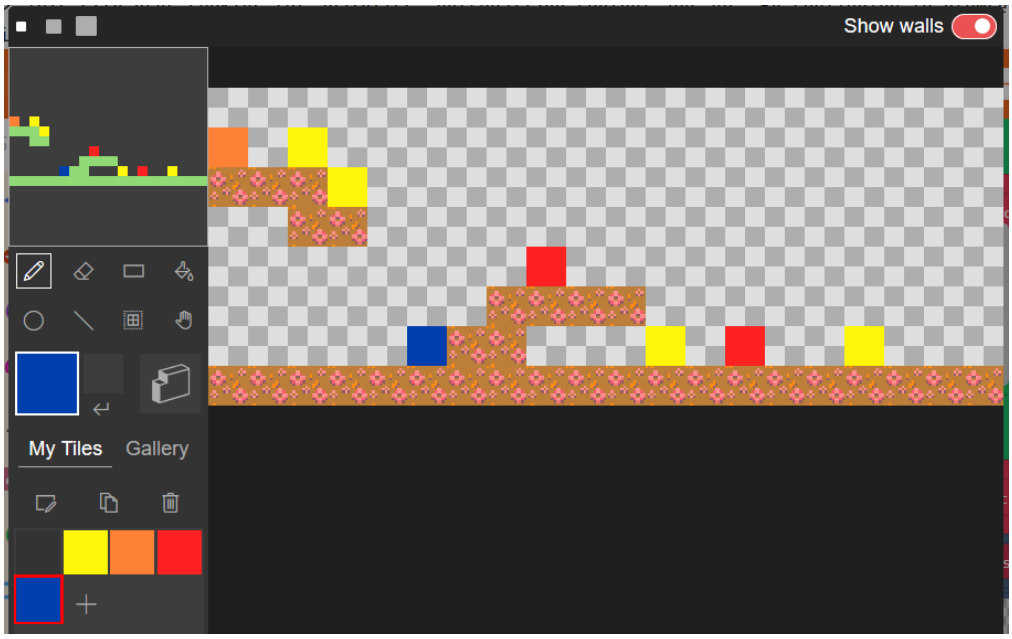
13.1 How to implement this Pattern in MakeCode

13.1.1 Step by Step instructions

A common pattern or mechanic in a game is to make the enemy move back and forward like a soldier on patrol. The easiest way to do this is to create wall blocks which the enemy bounces between. We can call these kinds of enemies bumpers.

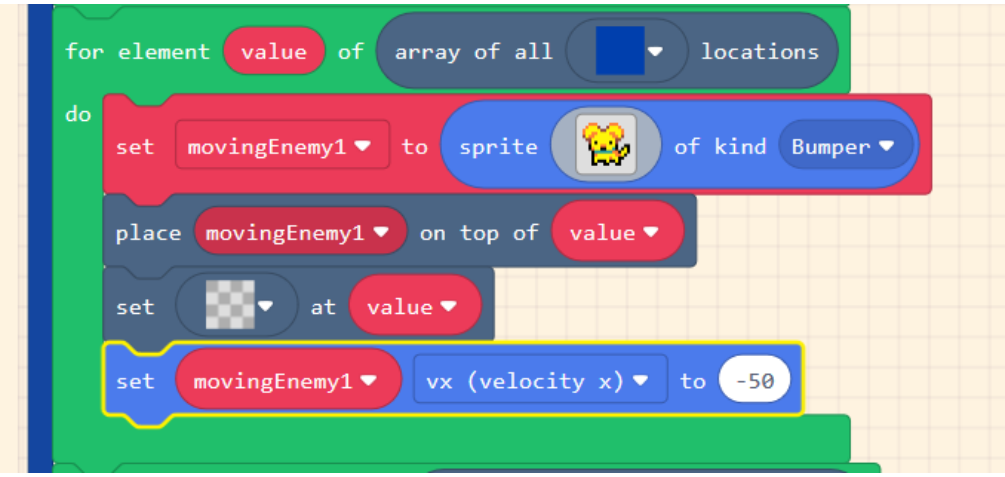
This tutorial assumes you have already added a static enemy pattern. We are going to add another kind of enemy in the same way. To do this duplicate the **for element** loop inside your **create level** function and add it back into the function.

We need to edit our tilemap and add a new colour of tile to our tilemap and place one in a location where it can bump between walls. In this example we will use a blue block.



add another tile

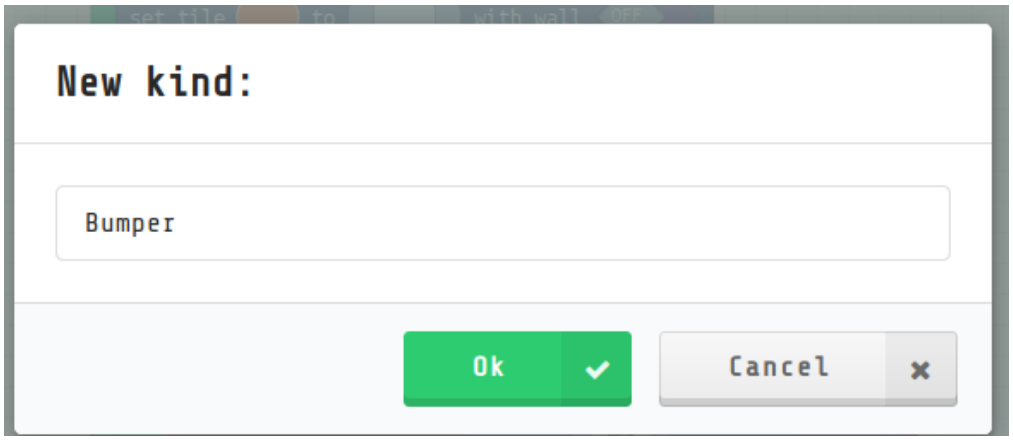
In our createLevels function, we are going to create a loop which finds all the blue blocks and does something with them. Follow the code example below to do this. You may recognise this pattern from the way that we create food in our game.



Change loop elements1

We need a new kind of sprite called a Bumper. To do this we'll have to make a new kind of sprite.

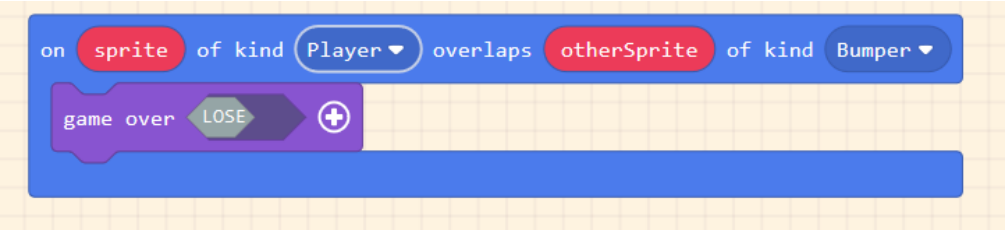
In the **Set My Sprite** to block click on the type of sprite and select **Add a new kind**.



patrolling enemies 1

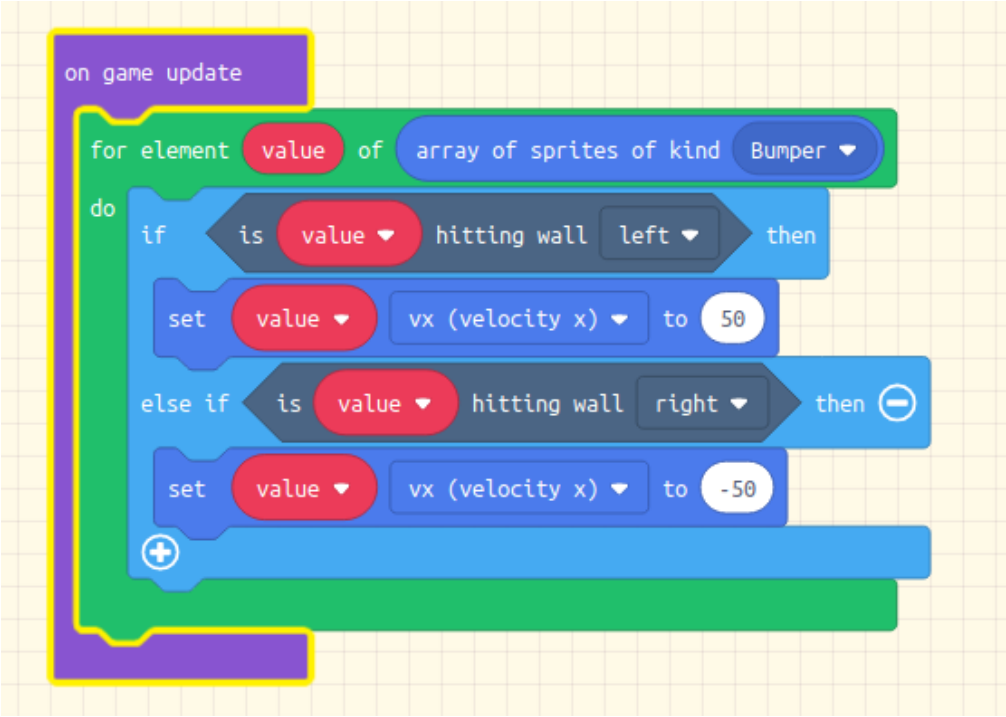
When this is done you can select it from the lists. We will also set our enemy heading off in a left or right direction. Set the velocity of our moving enemy to 50 (moving right) or -50 (moving left)

We need to set a Collision Listener so that the game is over if the player touches our moving enemy so create a Listener block as below.



Game over elements

Then we need to make our bumper enemy change direction when it hits a wall block. You can do this by copying the following block. You can see the green block is a familiar pattern where we loop through all the sprites of a particular type in this case.



patrolling enemies 2

13.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

To check that you are making the most of this pattern you can ask yourself the following questions:

- Are your blocks placed in the right place to make sure you enemies patrol in the right places
- Can place patrolling enemies towards the end of your game in a way that makes it tricky to get past them?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

This pattern may make your game much more challenging. To balance it out a potential next step may be to add the Jump on Enemies pattern to your game if you haven’t already.

14 Add Written Messages

mechanics space polish and systems

- **Name:** Add Written Messages
- **Description:** The game maker can display instructions and messages to the player at the start of the game. You can also add messages when something happens in the game. For example giving new levels a name.
- **Need for Pattern:** Having **written messages** is a way of giving playing instructions to the player at the start of the game, giving them clues as they progress or just making the game more fun to play by adding story elements.
- **Related Game Patterns:** Add Graphical Elements [related]
- **Coding Concepts involved:** Data, Events
- **Links to other Computing Patterns:** , Change Listener

14.1 How to implement this Pattern in MakeCode

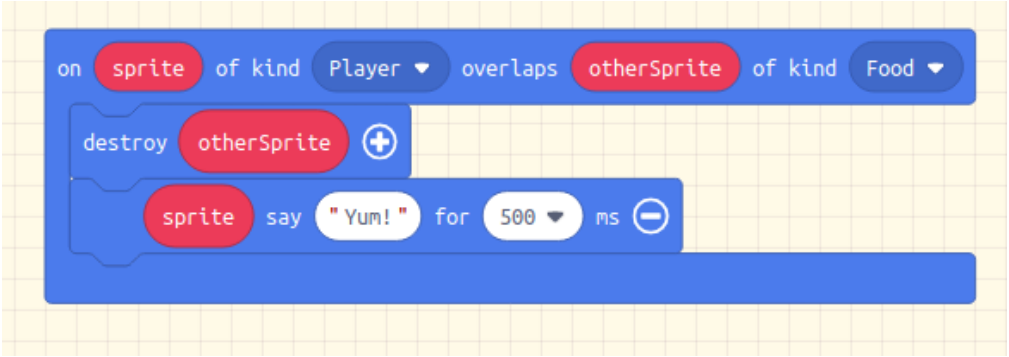
14.1.1 Step by Step instructions

Find the splash block under the Game section and drag one or more blocks to the beginning of the on start block.

add messages

You can also get your player to say something when an event happens. For example lets get the player to say Yum when collecting an apple.

Alter the overlap event code block to add in a sprite say block and set the time.

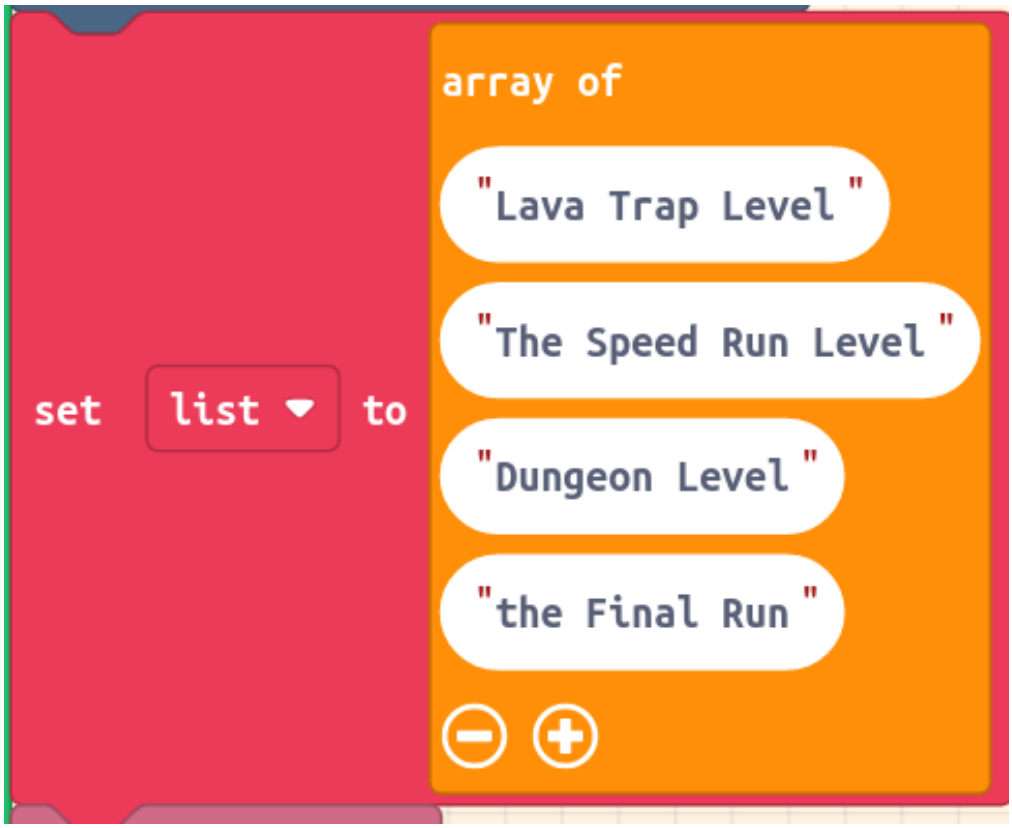


add messages

14.2 Extra Challenge

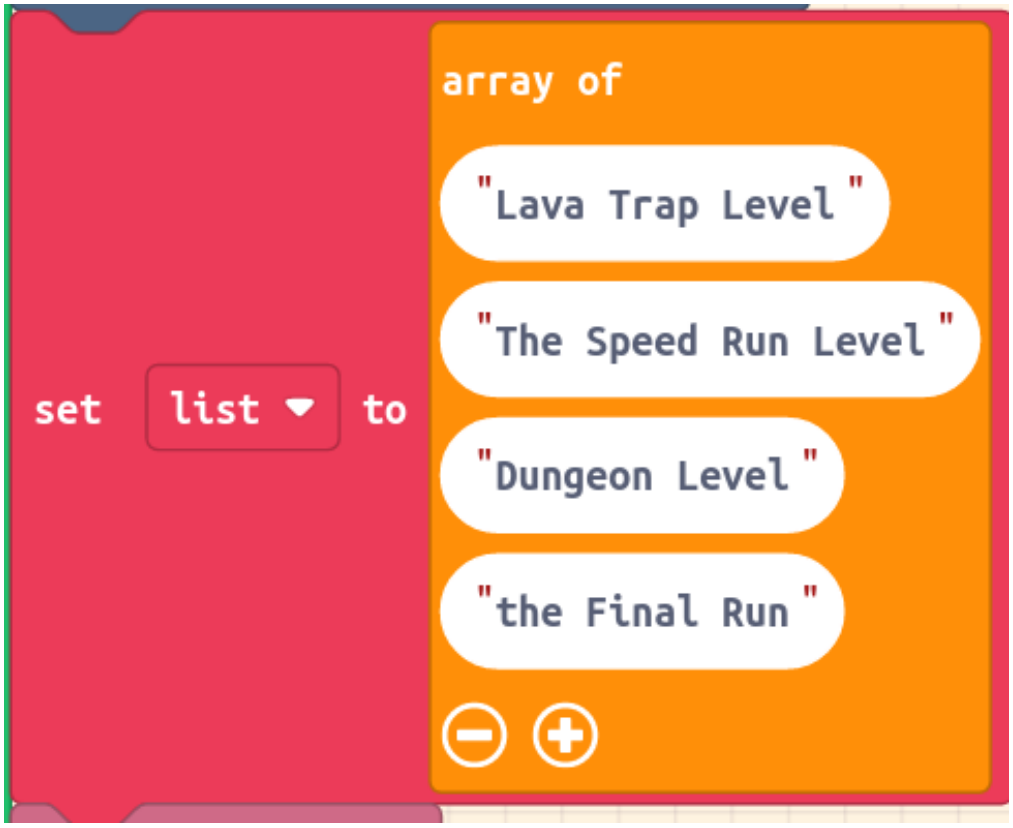
You can give your Levels name that come up before your players play them. To do this you will need to have added different levels to your game.

To start with we will create an Array (which is a kind of list) of the different names of the levels.



add a list

Now add a splash message at the start of your createLevel function. Note how you can use the level value to pick the right message from your list.



add a splash message

14.3 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example,

you can ask yourself the following questions:

- Check your messages aren't too long and disappear of the side of the screen
- Check that the messages appear in the right place

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

15 Animate Player

mechanics space polish and systems

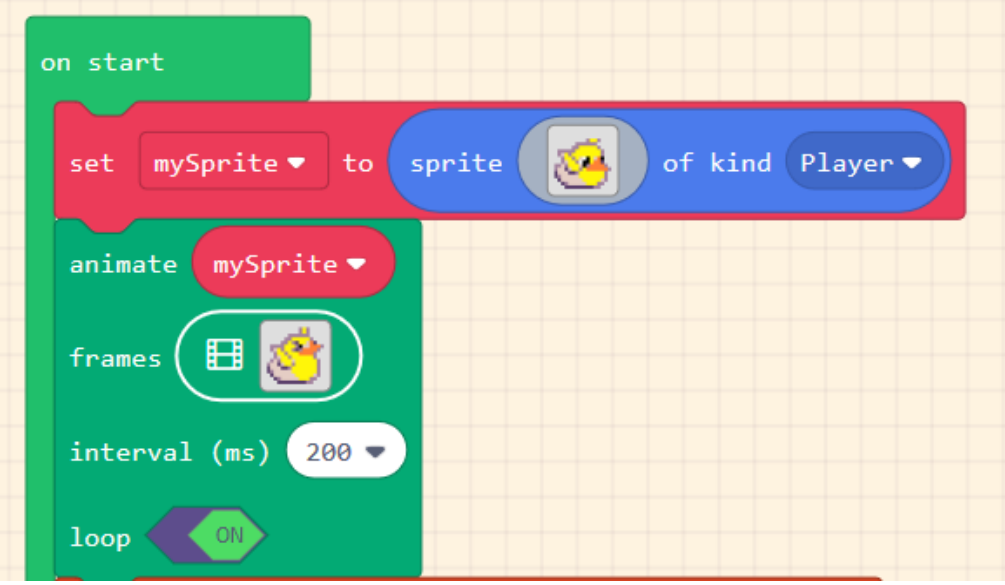
mechanics space polish and systems

- **Name:** Animate Player
- **Description:** The player is animated using different ‘frames’.
- **Need for Pattern:** This movement helps create a sense of dynamic movement and can help the game player’s immersion in the game.
- **Related Game Patterns:** Add Static Enemy [needed before adding this one], Jumping on Enemies [related]
- **Coding Concepts involved:** Data
- **Links to other Computing Patterns:** , Change Listener

15.1 How to implement this Pattern in MakeCode

15.1.1 Simple Animation of Player

We will create a simple animation using different frames of our player. To allow us to do this go to **Advanced > + Extensions > Add Animation** Drag in the animation block that you see below.



simple animation

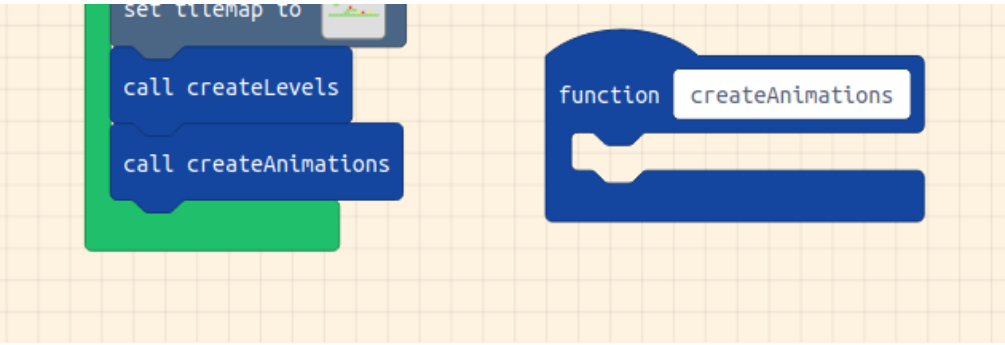
Click on the frames image to change the different frames of your animation, then interval to update how quickly the frames update and finally turn loop on so the animation keeps running.

15.1.2 Animated Walking and Jumping using Animation States

See the following https://makecode.com/_ecqJXwWrpJXw

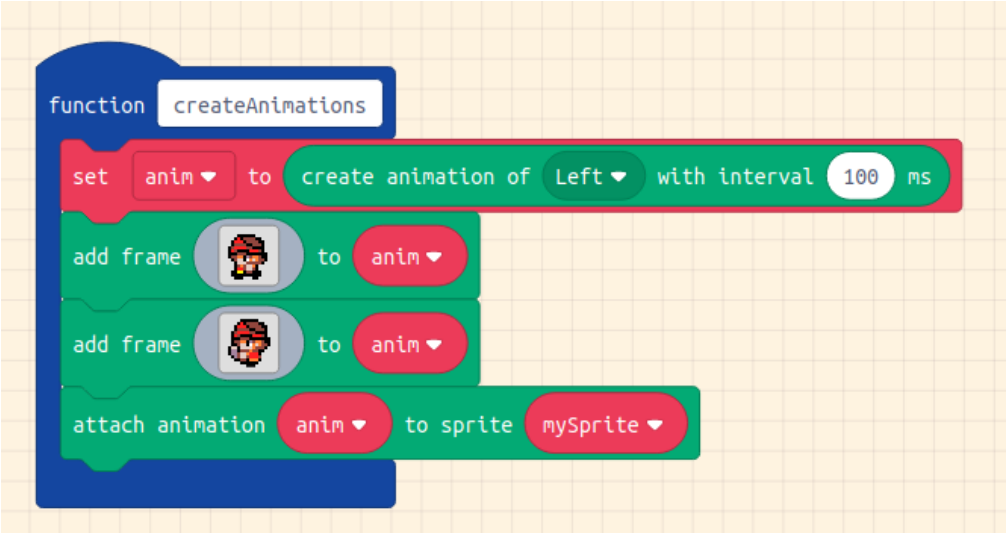
To create more authentic animations which change the direction of the player, show them walking and still when not moving we will use an animation with states. To start we will have three simple states moving **Left**, moving **Right** and not moving i.e. **Idle**

To start create an function to create our animation, call it createAnimations and then add a block to call it at the end of your on start loop.



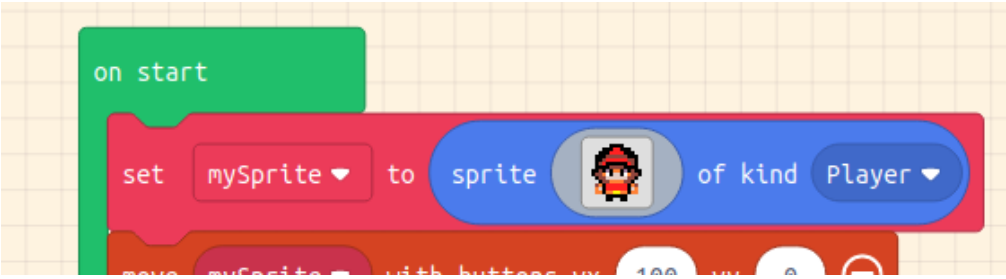
Now fill up the createAnimations block. To start add one animation which animates our player as then move to the left. To allow us to do this make sure you can add animation blocks go to

Advanced > + Extensions > Add Animation. Then add the blocks as per the screenshot below.

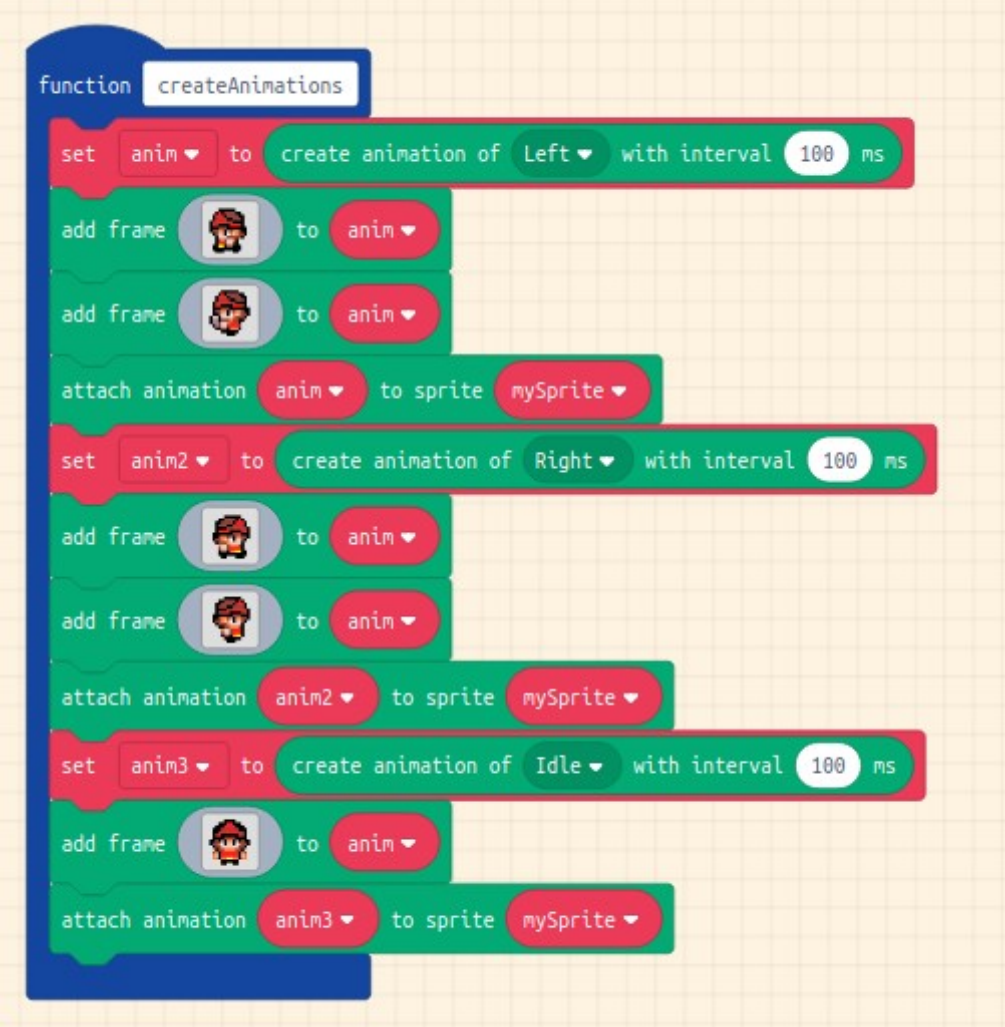


animate player states1

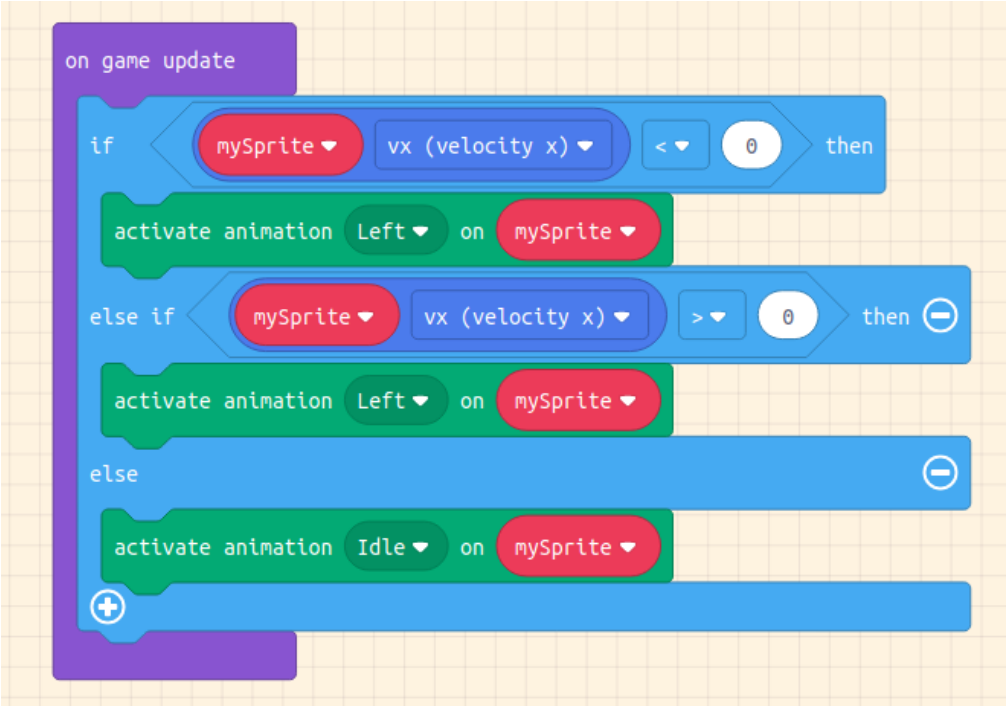
I have added simple two state animations by selecting different versions of a character from the Gallery. You can create your own animated player by drawing. Be sure to change your starting image too at the beginning of your on start loop



Now repeat the pattern for the moving right and not moving / idle states.



We have now to activate those animations create some code in a on **game update** loop that will be always running and listening for changes in conditions. We can use the players velocity in the x -axis to decide if they are moving left or right or not moving.



15.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

To check that you are making the most of this pattern you can ask yourself the following questions:

- Is your animation smooth? Do you need to add more frames?
- Do you want to add more states for jumping?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

16 Pattern Name

mechanics space polish and systems

- **Name:** Make Player Immune from Enemies
- **Description:** The player is not able to be zapped by enemies.
- **Need for Pattern:** Having **player immunity** is often needed if you have a system of lives. If you lose a life but don't move your player back to the start in a safe place, then the continuing overlap will cause you to lose all your lives at once.
- **Related Game Patterns:** Add Patrolling Enemy [required], Add Player Lives [required],
- **Coding Concepts involved:** Events
- **Links to other Computing Patterns:** , Change Listener,

16.1 How to implement this Pattern in MakeCode

16.1.1 Step by Step instructions

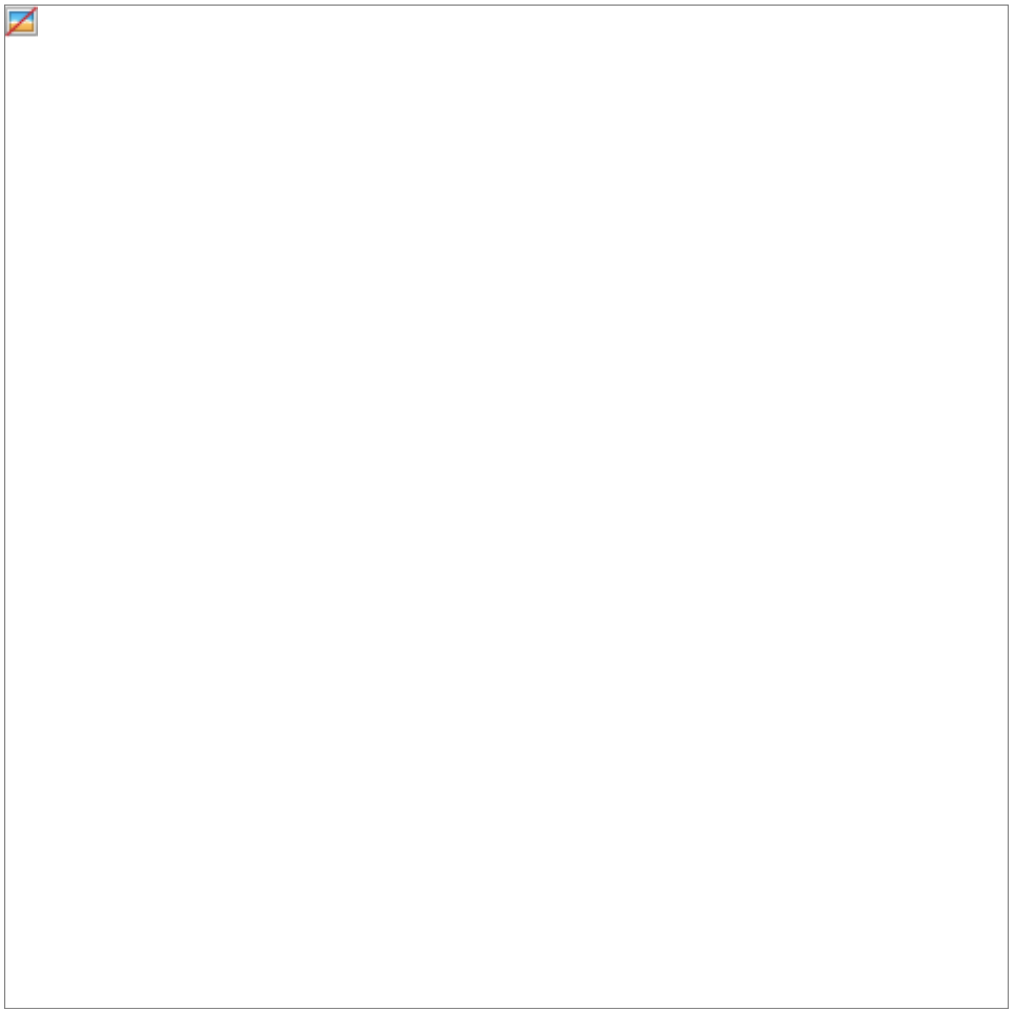
This pattern is needed when you have player lives and enemies. There is often a side effect

where a player can be overlapped with an enemy. The player needs to be made immune from enemies so that they cannot be harmed and they may also need to be moved. This tutorial assumes you have patrolling enemies and player lives patterns added to your game.

16.1.2 Add a delay and make Immunity obvious

In your **overlap listener** for the Bumper enemy type we will also play a sound to make it obvious that the player has been zapped.
We will also hange the look of the player sprite.

In the loop you will add in a **pause** when you lose a life. The **pause** means you cannot be zapped by players in this period.



make player immune

16.2 Test your Changes and Next Steps

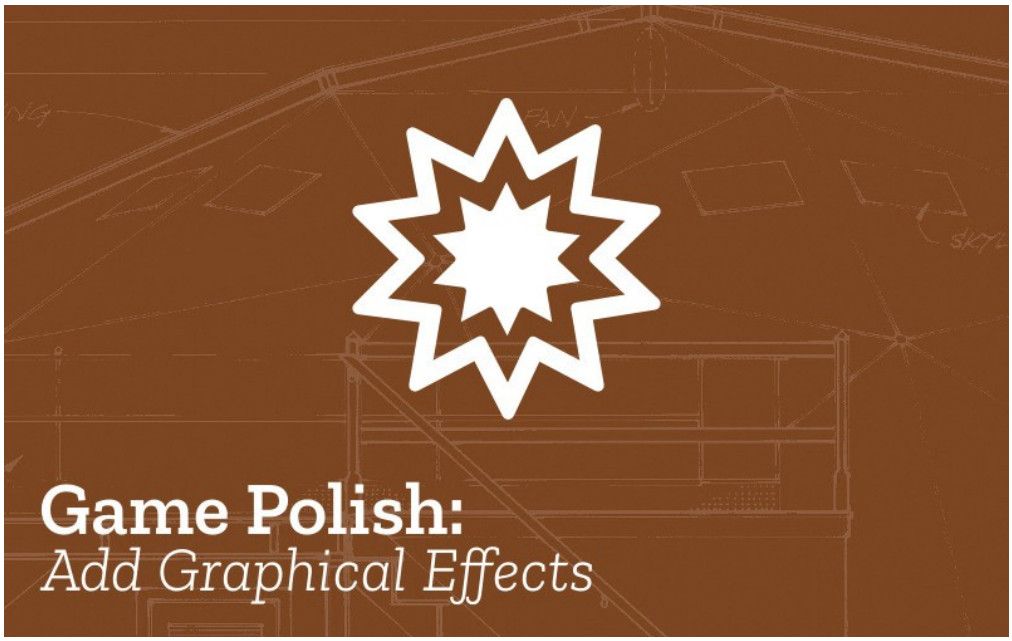
Test your game to check that your changes have the desired behaviour and that there are no side effects. For examples if you have more than one type of enemy, like simple static enemies, you will need to add this code in those overlap listeners too.

To check that you are making the most of this pattern you can ask yourself the following questions:

- Is the length of delay right or does it need to be longer or shorter?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

17 Simple Graphical Effects



simple graphical effects

- **Name:** Simple Graphical Effects
- **Description:** A common pattern is to add graphical effects when things happen in the game. For example the look of a player may change when they get zapped as a way of showing their death, or you can animate food when it is collected, or enemies when they get zapped.
- **Need for Pattern:** Having **graphical effects** is a way of increasing the excitement of the game and giving game players feedback on what happens in the game.
- **Related Game Patterns:** Jumping on Enemies [related]
- **Coding Concepts involved:** Events
- **Links to other Computing Patterns:** , Change Listener

17.1 How to implement this Pattern in MakeCode

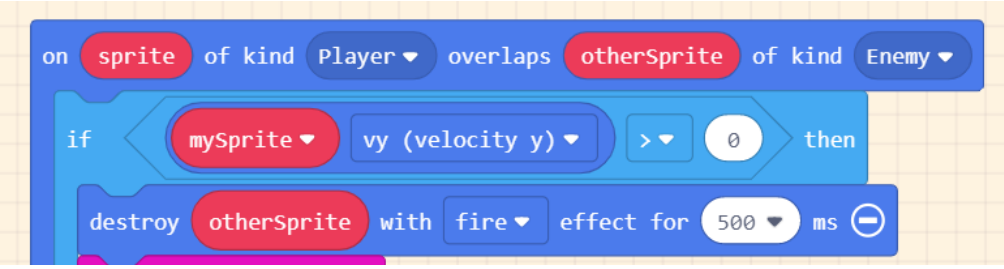
17.1.1 Animate Food Collection

Click on the plus sign next to the **destroy otherSprite** block and you can choose from many effects that happen to your sprite when the Food gets collected and how soon that effect happens. In the block below the confetti effects happens very quickly after the Food is touched.

Animate Food Collection

17.1.2 Animate Enemy getting Zapped

This requires you to have added the Jumping on Enemies pattern. In the **on sprite of kind Player overlaps otherSprite of kind Enemy** condition listener loop click on the plus sign to the **destroy otherSprite** block and you can choose from many effects that happen to your sprite when the Food gets collected and how soon that effect happens.



Animate Enemy getting Zapped

17.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

To check that you are making the most of this pattern you can ask yourself the following

questions:

- Have you animated all of the events that happen in your game?
- Are the timings right for your game or are they too long or short?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

A next step you might want to try is to add sound effects when things happen.

18 Add Sound Effects

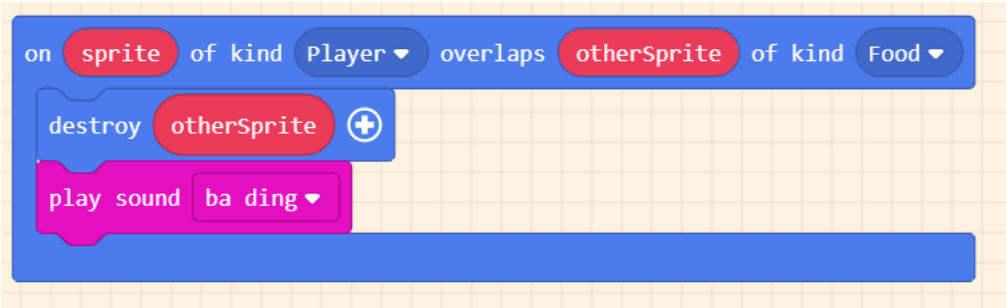
add sound effects

- **Name:** Add Sound Effects
- **Description:** A common pattern is to add sound effects when things happen in the game. For example sounds may play when a player gets zapped, or when food when it is collected, or when enemies are zapped.
- **Need for Pattern:** Having a **sound effects** is a way of increasing the excitement of the game and giving game players feedback on what happens in the game.
- **Related Game Patterns:** Jumping on Enemies [related]
- **Coding Concepts involved:** Events
- **Links to other Computing Patterns:** , Change Listener

18.1 How to implement this Pattern in MakeCode

18.1.1 Sound Effect for Food Collection

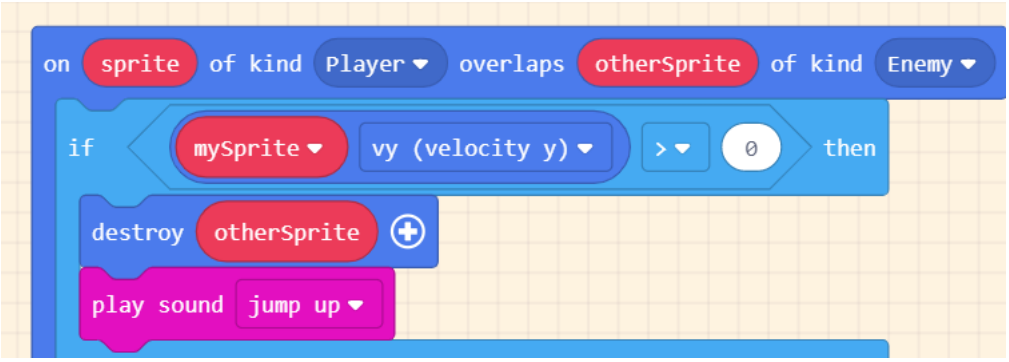
Click on the plus sign next to the **destroy otherSprite** block and you can choose from many effects that happen to your sprite when the Food gets collected and how soon that effect happens. In the block below the confetti effects happens very quickly after the Food is touched.



Sound Effect Food Collection

18.1.2 Sound Effect for Enemy getting Zapped

This requires you to have added the Jumping on Enemies pattern. In the **on sprite of kind Player overlaps otherSprite of kind Enemy** condition listener loop add a block which plays a sound effect. You can experiment to see which one you like best.



Sound Effect Enemy getting Zapped

18.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

A next step you might want to try is to add a sound track.

19 Add Sound Track

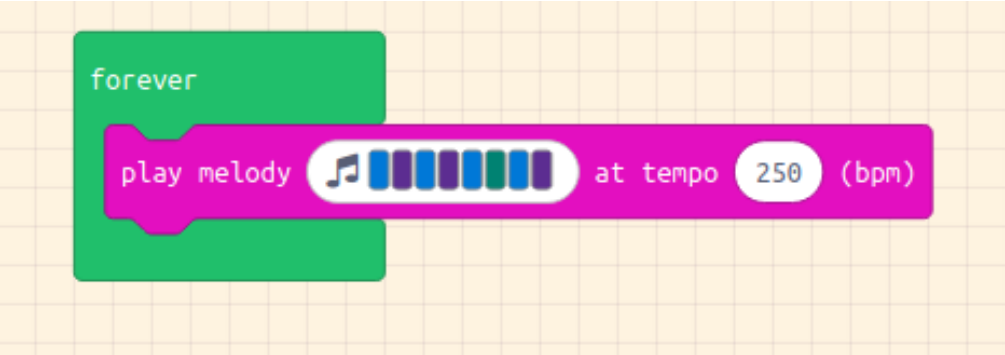
add sound track

- **Name:** Add Sound Track
- **Description:** A common pattern is to add sound track or background music to the game.
- **Need for Pattern:** Having a **sound track** is a way of increasing the excitement of the game and communicating the feeling of the game to the players.
- **Related Game Patterns:** Add Sound Effects [related]
- **Coding Concepts involved:** Events

19.1 How to implement this Pattern in MakeCode

19.1.1 Sound Track for One Level

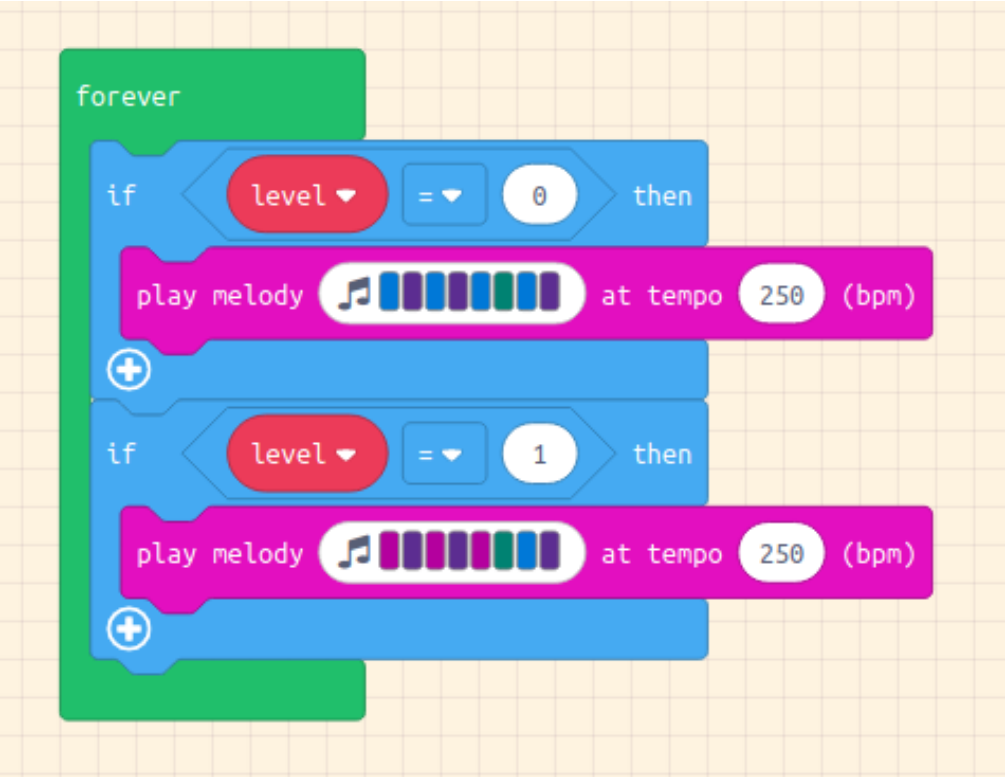
If you want to keep the same music looping or if you only have one level then you can add a soundTrack in a simple way using the code blocks shown below.



Sound Track

19.1.2 Sound Effect for Enemy getting Zapped

If you already have more than one level then you may want to change the tune for different levels. You can see how to do this in the screenshot below.



Sound Track

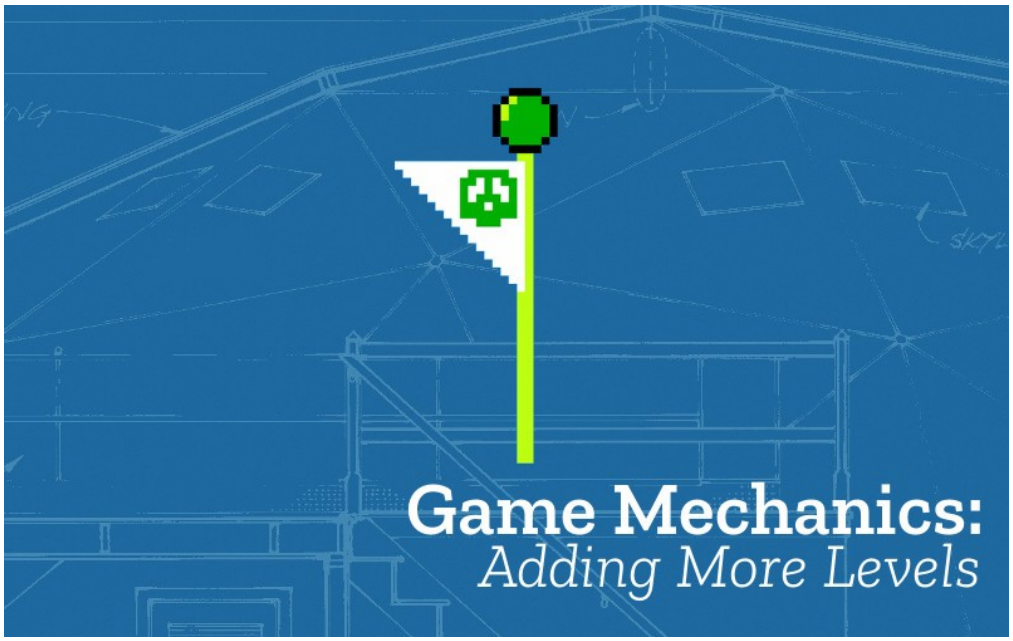
19.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

A next step you might want to try is to add sound effects when things happen.

20 Add More Levels



mechanics space polish and systems

- **Name:** Add More Levels
- **Description:** The player reaches the goal of the first Level of the platformer and then progresses to the next level (and beyond)
- **Need for Pattern:** Having more than one level is a way increasing the length and challenge of the game. You can also bring other elements of design and even game mechanics to new levels.
- **Coding Concepts involved:** Data, Variables,
- **Links to other Computing Patterns:** Systems Dynamics, Change Listener

20.1 Putting the Pattern into Practice

20.1.1 Create a “level” variable

Now we will reate a variable to hold the number of level we are on. We will then be able to change it. We create this level variable at the end of our ***on start*** loop.

Create Variable

Add set Variable

20.1.2 Create a new chooseLevel Function

Create a new Function by clicking on the **Advanced** tab on our toolbar, then **Functions** And then click **Make a Function**. Enter chooseLevel in the white box.

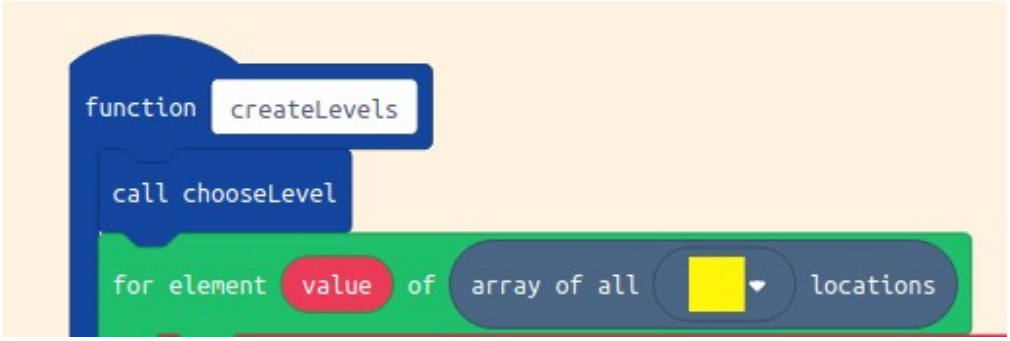
Add set Variable

20.1.3 Move tilemap to new Function

Drag only the **set tilemap to** into the new **chooseLevel** function by holding down **Control** on your Keyboard while you drag the block. Right click and select **Format Code** to make the screen tidy.

20.1.4 Add a link to the new function

Drag in the **call chooseLevel** from **Functions** to the end of the begining of the **createLevel** function.



game level four

20.1.5 Create the Logic switcher for the Level Design

Now add a Logic block **if true then** inside the **chooseLevel** block Move the the **set tilemap to** inside the logic block. In the new **if true then** block replace **true** with at **0 = 0** block from Logic section.

Right click on the Logic block **if level = 0** and duplicate it. Change 0 to 1 in the second block and drag it back into the **chooseLevel** function.

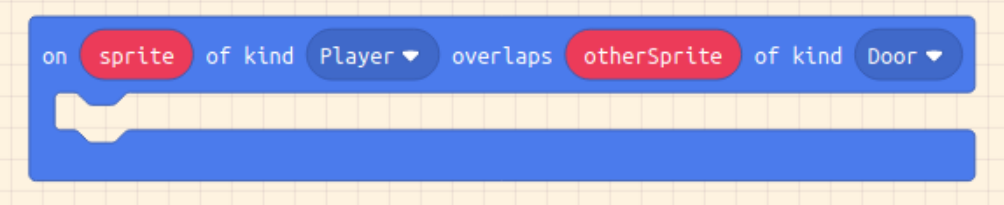
Repeat the process again changing the **level** number to two. Move a **game over - win** block into to this gap and delete the **tilemap** block.

game level four

20.1.6 Change the overlap (change) Listener between Player and Goal Sprite

In our game we have a listener which is always checking to see if there is an overlap between our Player and the Goal Sprite.

In our starting template this lists only a Game Over - Lose block. Delete that block so the on overlap listener block is empty.

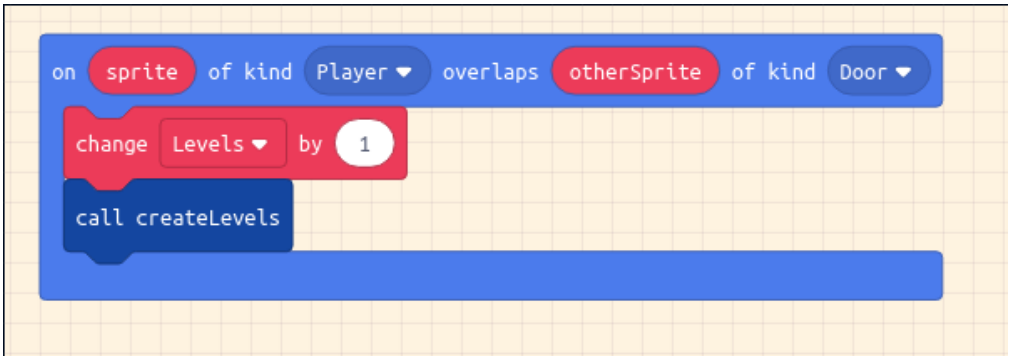


Remove Game over Block

We will replace it with code that changes the level number and builds a new level and moves the player back to the starting point.

20.1.7 Code what happens when we complete a level

In the **on sprite of kind Player overlaps Door** listener block add a **change level by 1** block. Next add call createLevels block to recall the function that does the work of adding the different blocks to the game.



insert change levels block

20.1.8 Remove any thing that might remain from the last level

If you are not careful then some of the things from the last level like food or enemies may carry over to your next level.

From the **Sprites** area drag in a **destroy all sprites of kind xx** block to the start of the **createLevel** function. Change the option to **Food** or **Enemy** or what ever other kinds of object you may need to clear for the previous level.

Add set Variable

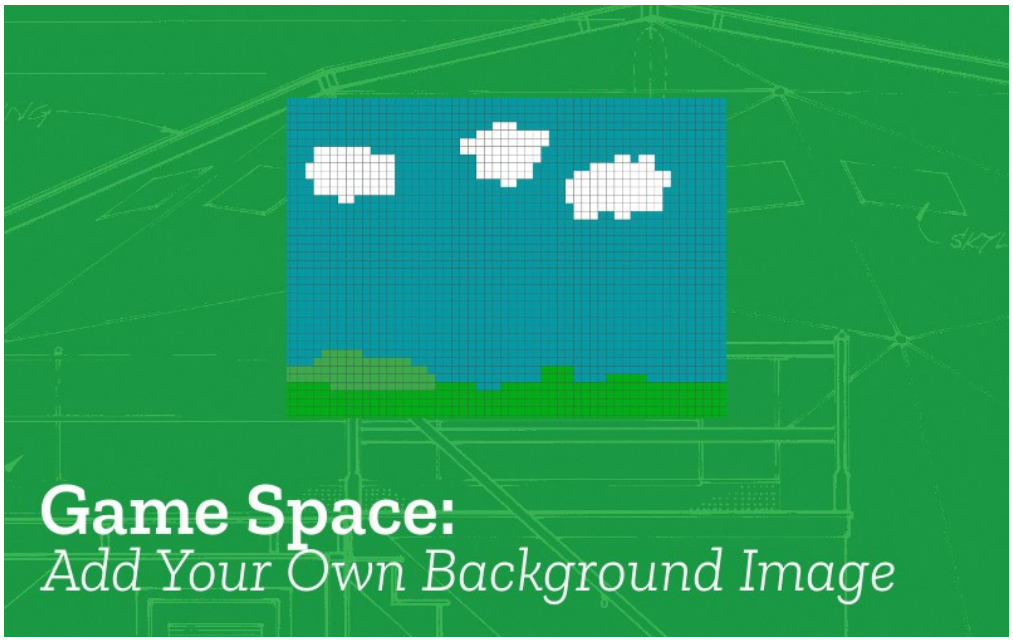
In the image above all previous items of food are removed. You may need to several of these kinds of blocks for different types of items you add to your game.

20.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example check that each time you touch the end goal you progress by a level and the design matches.

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

21 Change Background Image



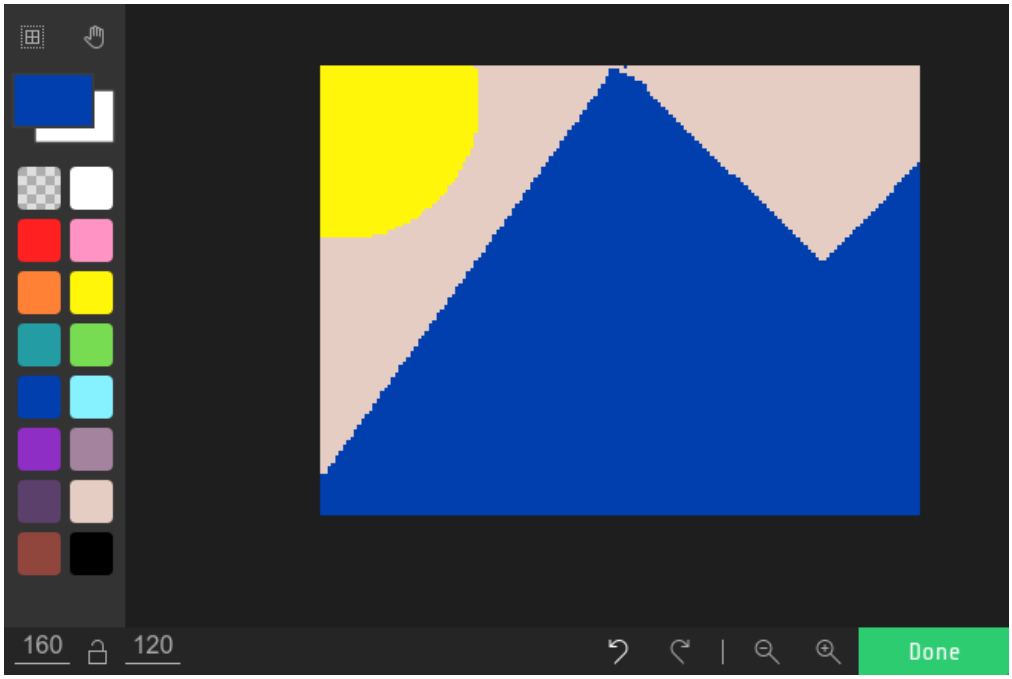
mechanics space polish and systems

- **Name:** Change Background Image
- **Description:** This pattern gives you the ability to add a drawn image as the background of your game.
- **Need for Pattern:** Having **a designed background images** is a way of increasing the player's immersion in the game and their sense of connection to the story.
- **Related Game Patterns:** Add More Levels [related]
- **Coding Concepts involved:** Data

21.1 How to implement this Pattern in MakeCode

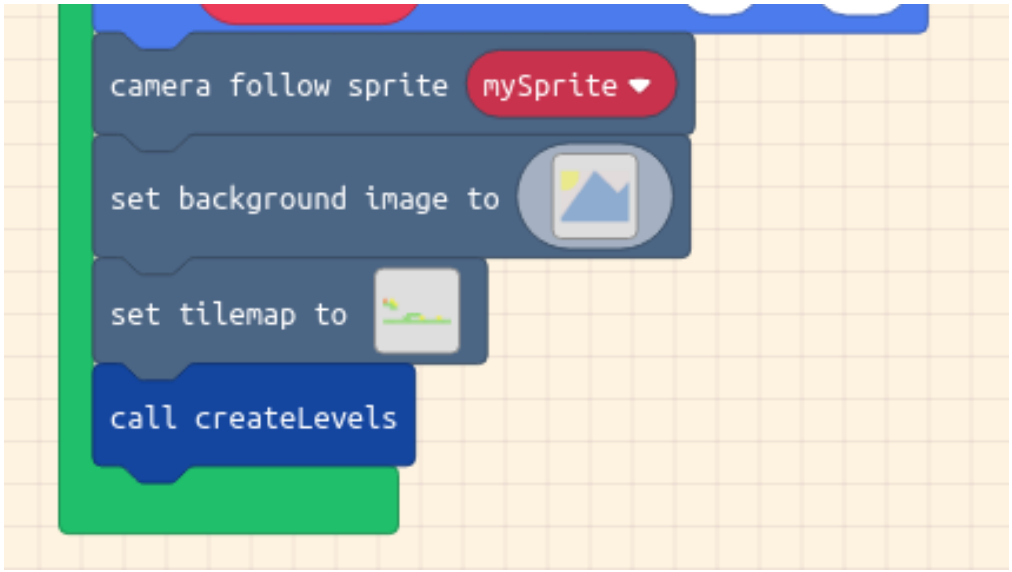
21.1.1 Change the background image

Drag in a **set background image to** block to replace the **set background color** block



change level shape

Then click on the square image and using the tools draw a background and click on done.



change level shape

21.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example,

you can ask yourself the following questions:

- Does the colour of your Background interfere with any of your other game elements?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

22 Change Level Design

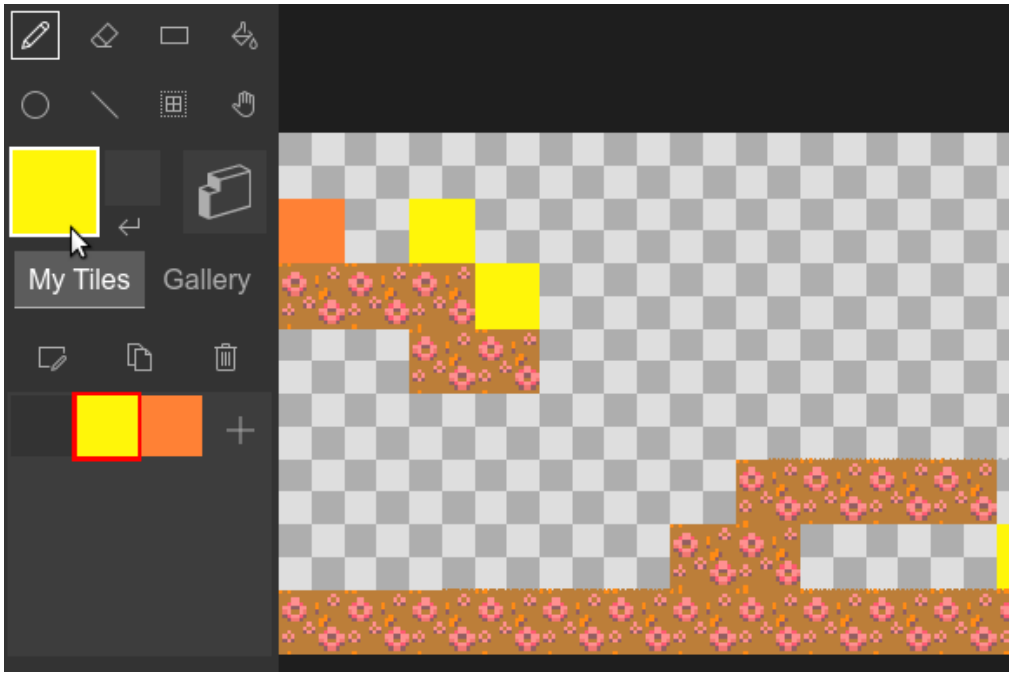
Change Level Design

- **Name:** Change Level Design
- **Description:** You may want to change the location of the platforms or add more food to collect or change the location of your end sprite block.
- **Need for Pattern:** Having the ability to change the level space design is important to be able to change the challenge level of your game.
- **Coding Concepts involved:** Lists,
- **Links to other Computing Patterns:** Separate Formatting from Data

22.1 How to implement this Pattern in MakeCode

22.1.1 Open the Tilemap Editor and add New Food Items

Click on your tilemap image level 0 for and add in more food items by clicking on the yellow square in My Tiles and then clicking on squares in your design on the main window.



Change Level Design

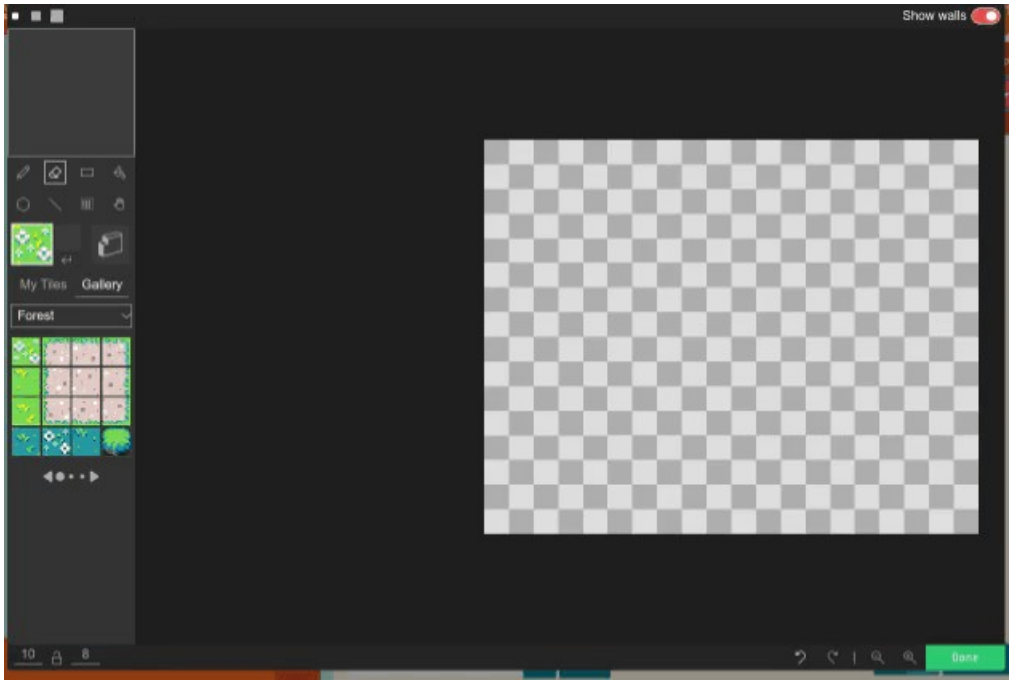
When you click **Done** you will see and can test the updated design in the preview window on the left.

You can remove Food items by clicking on the eraser (second icon) in the Tile Map editor and clicking on existing Yellow sqaures.

22.1.2 Use the Tilemap Editor and add in new Platforms

To add or change platforms. Click on your tilemap image and add in more platforms using the tile gallery. Create a line of tiles to be a platform. Choose a suitable tile for your platform. You can choose the first, default one or another kind if you want to create variation.

The program does not know that we want these squares to be ones our player can stand one and not travel through yet. To do this select the **Wall Drawing tool** and click the same squares. Then click on Done.



Create a row of images and walls

22.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example

you can check the following:

- Check your levels aren't too easy or to tricky

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

23 Change Level Shape

mechanics space polish and systems

mechanics space polish and systems

- **Name:** Change Level Shape
- **Description:** You may want to make your level longer or you may want to change the shape of one or more levels completely so they have to jump up instead of travelling from left to right.
- **Need for Pattern:** Having **different shaped levels** is a way of making the game seem more exciting to explore. It may also change the nature of your game especially if you make your player climb up or fall down a level which is tall and thin.
- **Related Game Patterns:** Add More Levels [related]
- **Coding Concepts involved:** Data, Events
- **Links to other Computing Patterns:** , Change Listener

23.1 How to implement this Pattern in MakeCode

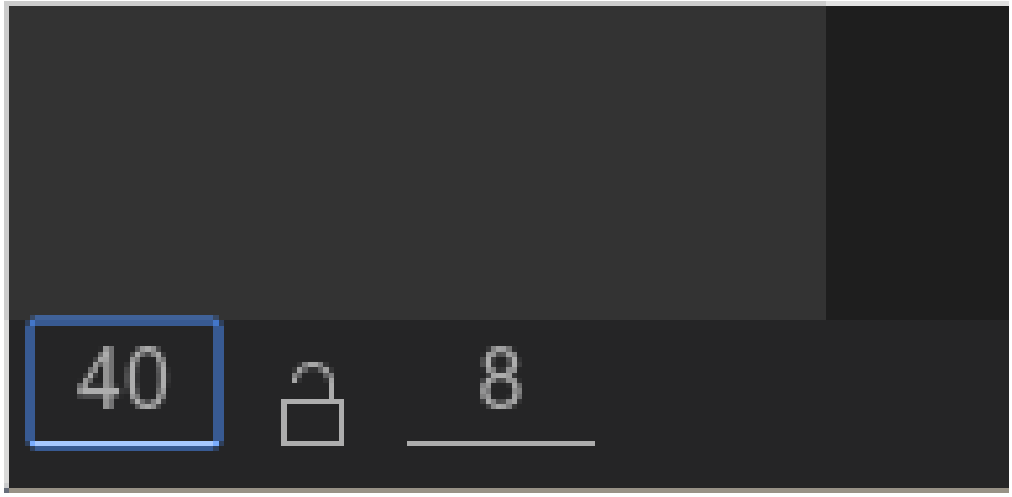
23.1.1 Change the width of your level to make it longer

The most simple way to add this game pattern is to make your level length longer. To do this click on your tilemap image and click on the numbers at the bottom left to change them.

The first one is the width so try changing that from 20 to 40. This make the level twice as long. Fill in the blank space to add your level design.

23.1.2 Change the shape of your level.

Another way to radically change the feel of your game is to make it tall and thin. Try this out by making the second value (the height) much higher than first value (the width)



change level shape

If you want your player to start at the bottom then you'll need to change your starting position. You may need to do some maths. The position is in pixels. If each square on the tilemap is 16 pixels and it is 40 squares high then the screen will be 40 x 16 pixels high. You can set your player position based on that.

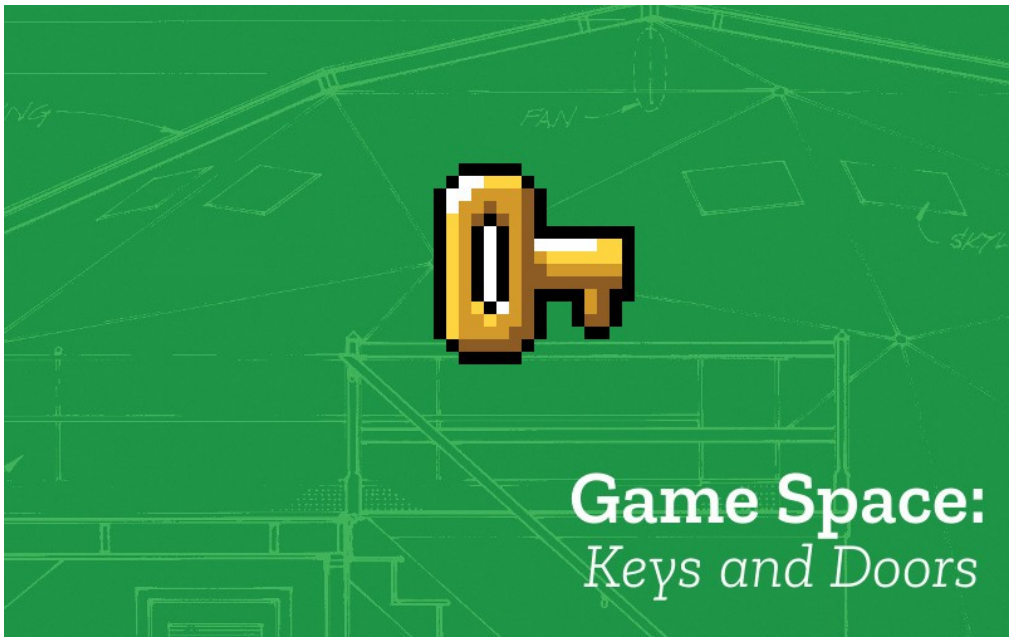
23.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example,
you can ask yourself the following questions:

- Check your levels aren't too long and therefore feel boring

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

24 Key and Door



keys and doors

- **Name:** Key and Door
- **Description:** The player needs to collect a key before they can exit the level of the platformer game.
- **Need for Pattern:** Having a **Key and Door** is a common pattern which adds to a sense of exploration to the game. You can use this in combination with using points to collect other objects.
- **Related Game Patterns:** Add Levels [suggested before adding this one], Collect Points [related]
- **Coding Concepts involved:** Data, Loops, Events
- **Links to other Computing Patterns:** , Change Listener, Input Event, Systems Dynamics

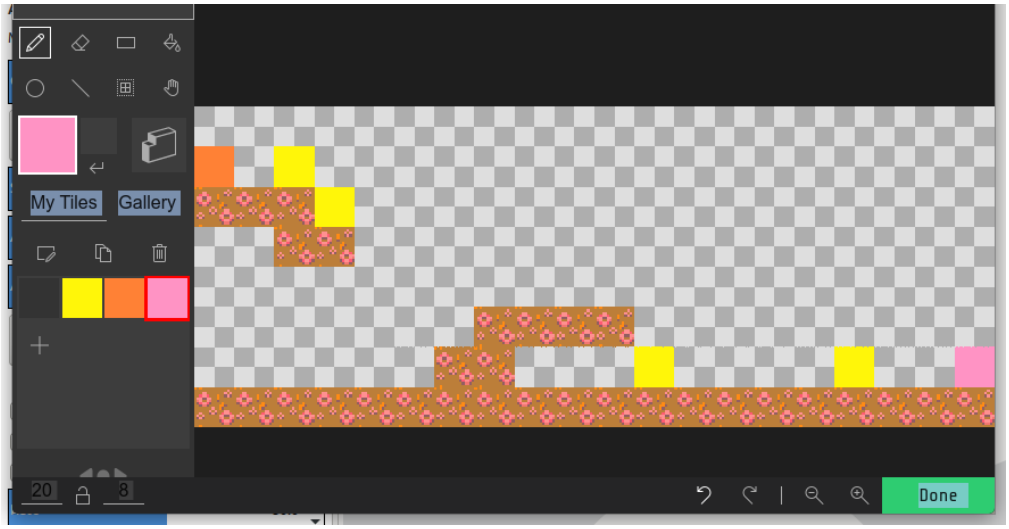
24.1 How to implement this Pattern in MakeCode

24.1.1 Step by Step instructions

In this simple application of the key and door pattern we will keep our current end goal, which in our starting template was the chest, and add in a new kind of sprite called a key which the player needs to collect before touching the end goal has an effect.

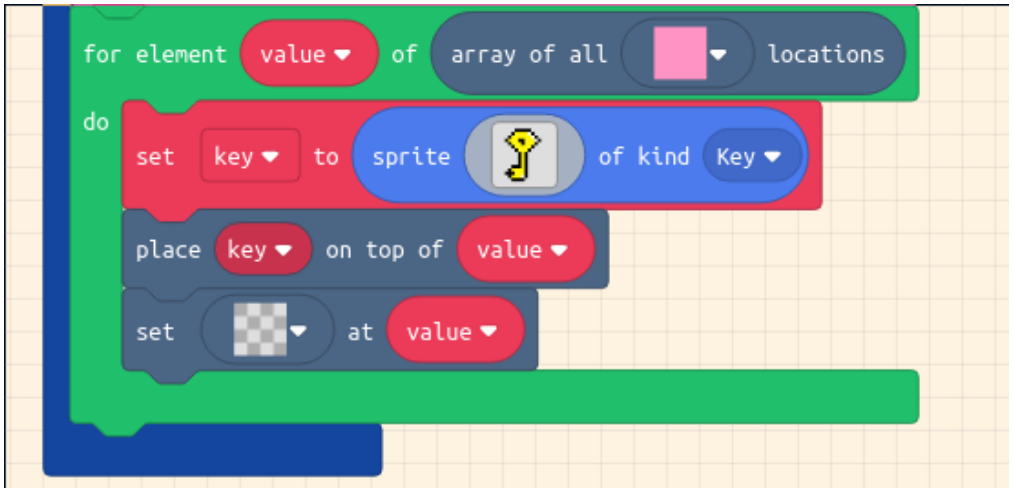
First add in a new tile colour to represent our key in your Tilemap plan of the first level. There are more detailed steps on how to do this in the add Static Enemy instructions. But in short create one tile to represent your location of your key.

To do this click on your tilemap to bring up the editor. Click on **My Tiles > + (plus sign)** and fill this new tile entirely with one colour. Then select that tile and place one tile in your level design where you want your key to be. This is shown in the screenshot below.



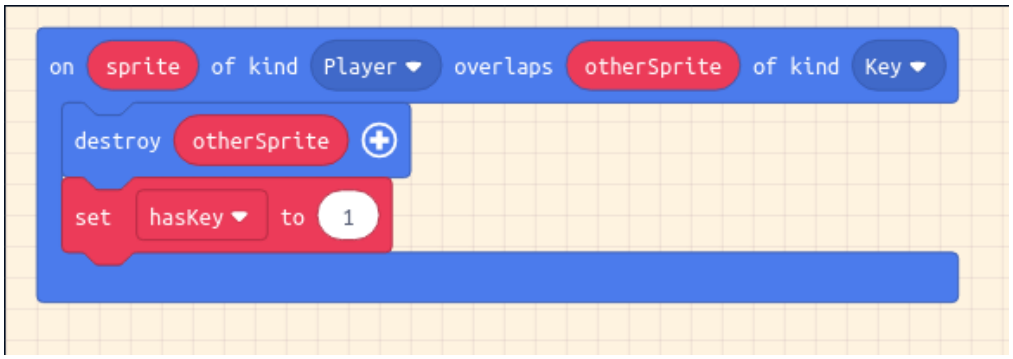
key and door 1

Follow the same kind of patter used in adding food to the level by adding a for element loop to look for the new coloured squares. This is explained in more detail in adding a static enemy pattern. Change the names of the variable and add in a new sprite type called Key. Change the image by drawing a key too. This is shown in the screenshot below. You can change the image of your end goal to a door also if you want to.



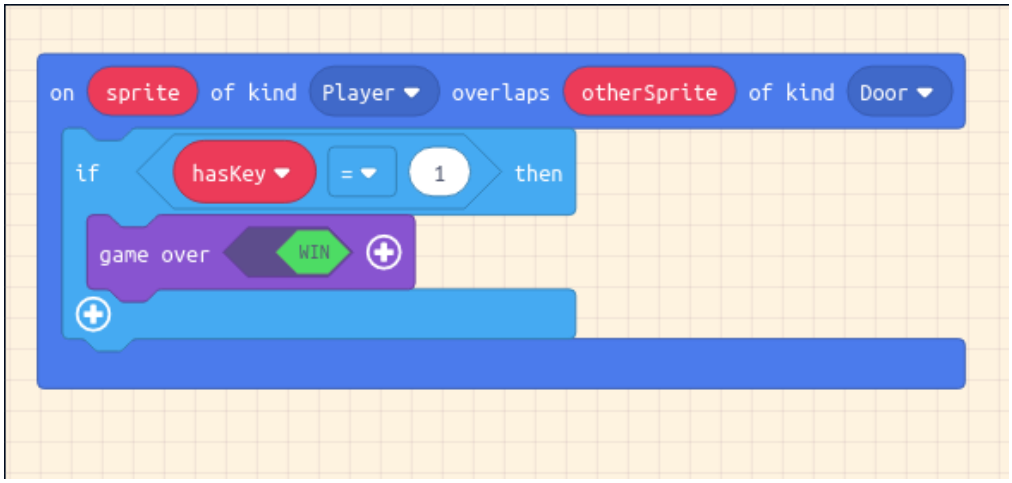
key and door

Add in a condtion listener that collect the key when the player touches it. We need to know if the player has the key so we will create a new variable called **hasKey** We will set this to 1 to show that the player has one key.



key and door

Then only progress if the player is touching the end goal and the **hasKey** variable has been change to one, as in the screenshot below.



key and door 4

24.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects?

To check that you are making the most of this pattern you can ask yourself the following questions:

- Is one key enough to collect? If you want to add keys more how would you change the code to make sure your program behaves the way you want it o.

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

25 Remixing a Platformer Game - Power Up Higher Jump

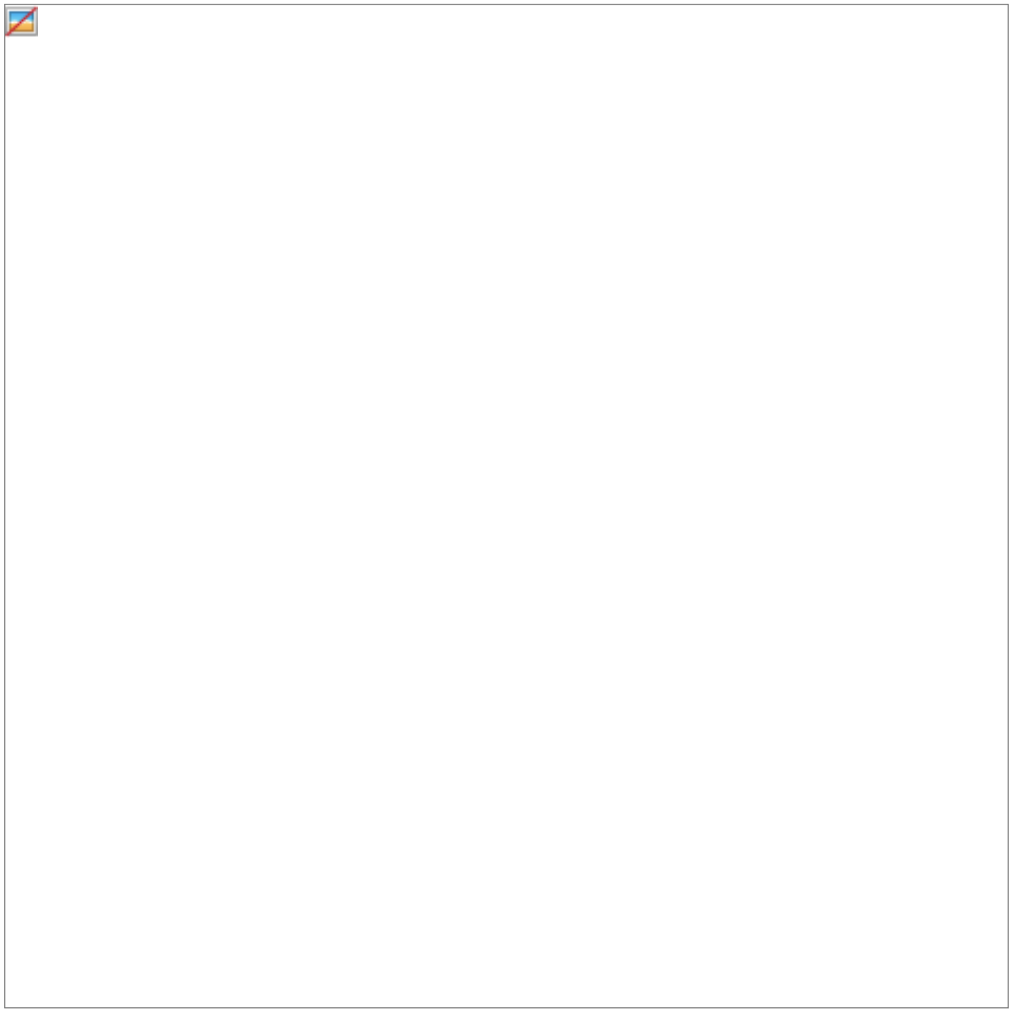
power up higher jump

- **Name:** Power Up Higher Jump
- **Description:** The player is able to jump higher if they collect a token that acts as a way of powering up their abilities.
- **Need for Pattern:** Having a **Power Up Higher Jump** is a way of increasing the interest and challenge of our game. The designer is able to create areas of the game that the player can only get to if they collect the power up.
- **Related Game Patterns:** Add Static Enemy [related]
- **Coding Concepts involved:** Data, Loops, Events
- **Links to other Computing Patterns:** , Change Listener, Input Event, Systems Dynamics

25.1 How to implement this Pattern in MakeCode

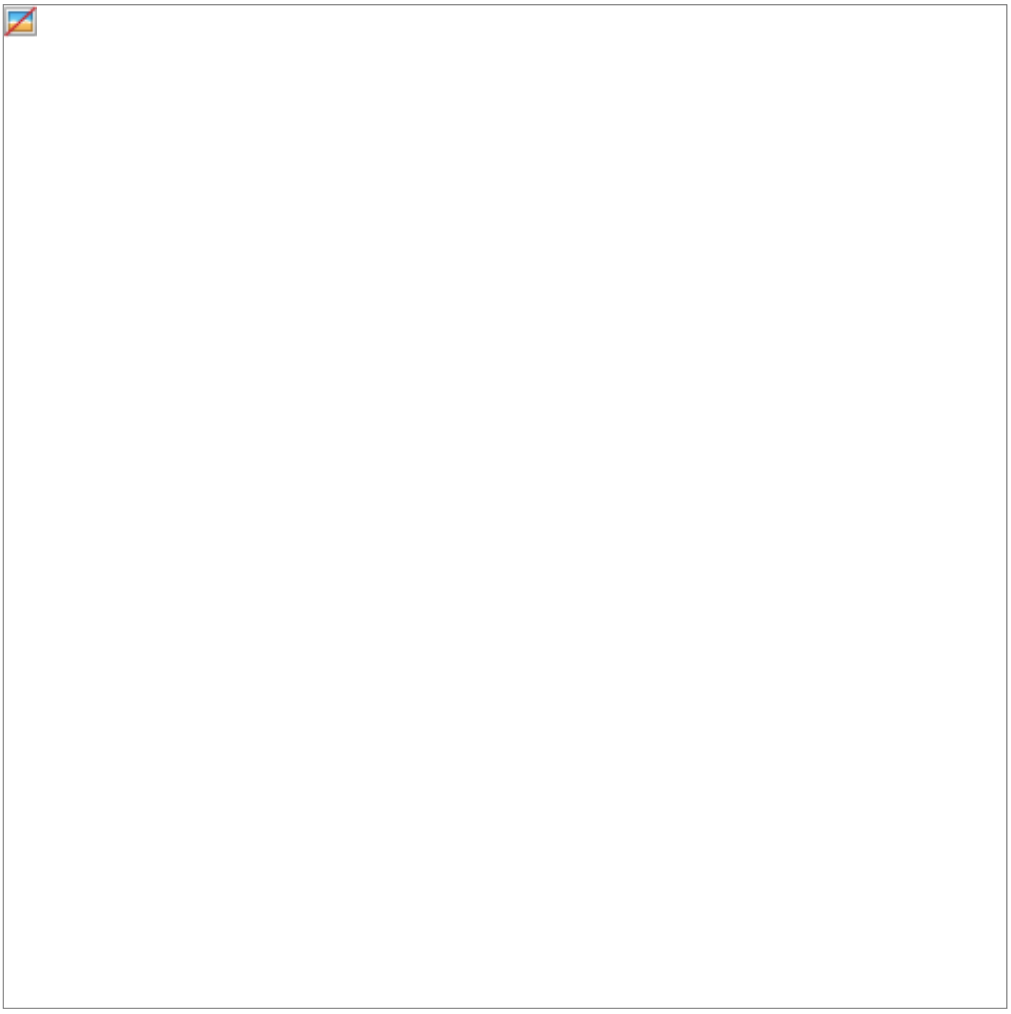
25.1.1 Create our power up sprite

To create a power up we will replicate the for element loop used to display the Food elements on our game. So the first step is to duplicate that and to make changes to the values. You can find more detailed instructions on how to do this is the create static enemies pattern. To start change your level Tilemap design to add a new colour tile to act as our power up and change the design to include an area that the player would not be able to access with their normal jump.



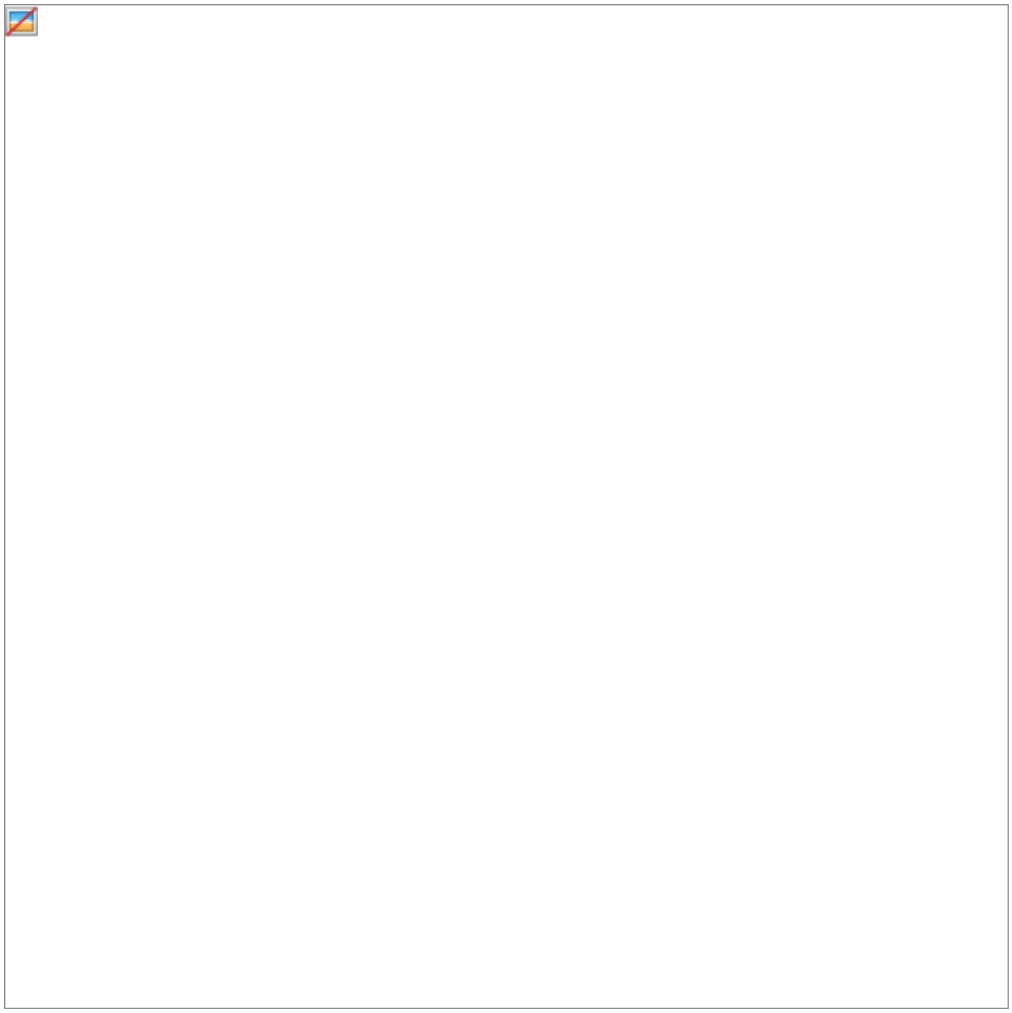
power up higher jump

Follow the code pattern shown in the screenshot below to make the power up appear in our

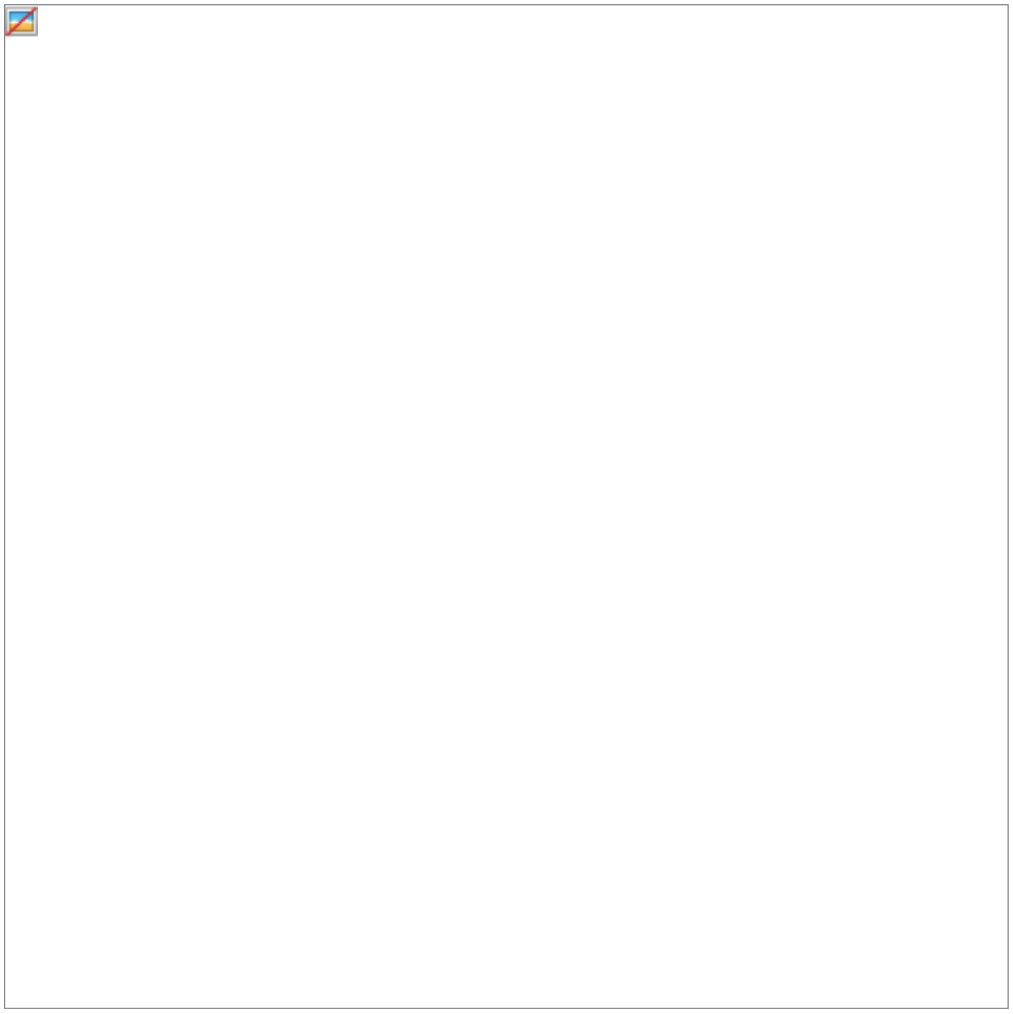
game. 

25.1.2 Create a new Variable

To be able to change the jump height of our player we will need to create a new variable for our game. This will then be used when the player presses the jump button. To do this create a new variable called jumpVelocity and and set it to the value you have in your on **A button pressed** input listener.



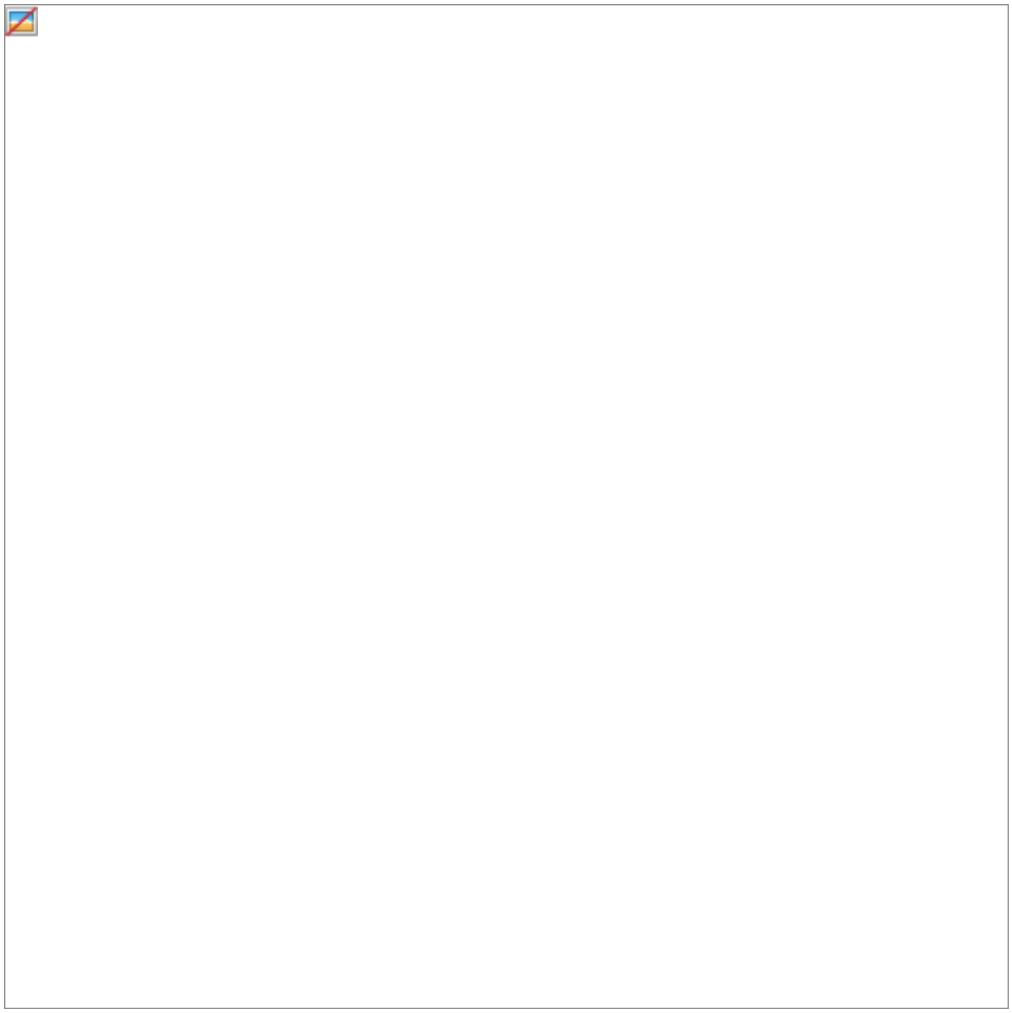
Change Input Listener You can now replace the original velocity y (vy) value with this new variable.



power up higher jump

25.1.3 Create the new Variable Value

Then we will add a condition listener to see when the player is overlapping this power up. When they do we can make the power up dissappear and increase the velocity of the players jump. By increasing the value of the variable that we have just created.



power up higher jump

25.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

To check that you are making the most of this pattern you can ask yourself the following questions:

- Can you jump up to your target area only after you collect your power up?
- Maybe you can make it tricky to access some areas when you have a higher jump, this increases the challenge by making the player be careful about the order they do things.

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

26 Collect all Food before Progressing

mechanics space polish and systems

- **Name:** Collect all Food before Progressing
- **Description:** The player must collect all items of food before progressing.
- **Need for Pattern:** Having our player **Collect all Food before Progressing** is a way of increasing the challenge of a game. It works well in combination with having a timer.
- **Related Game Patterns:** Add a Time [related], Add more Levels [related]
- **Coding Concepts involved:** Data,
- **Links to other Computing Patterns:** , Change Listener ## How to implement this Pattern in MakeCode

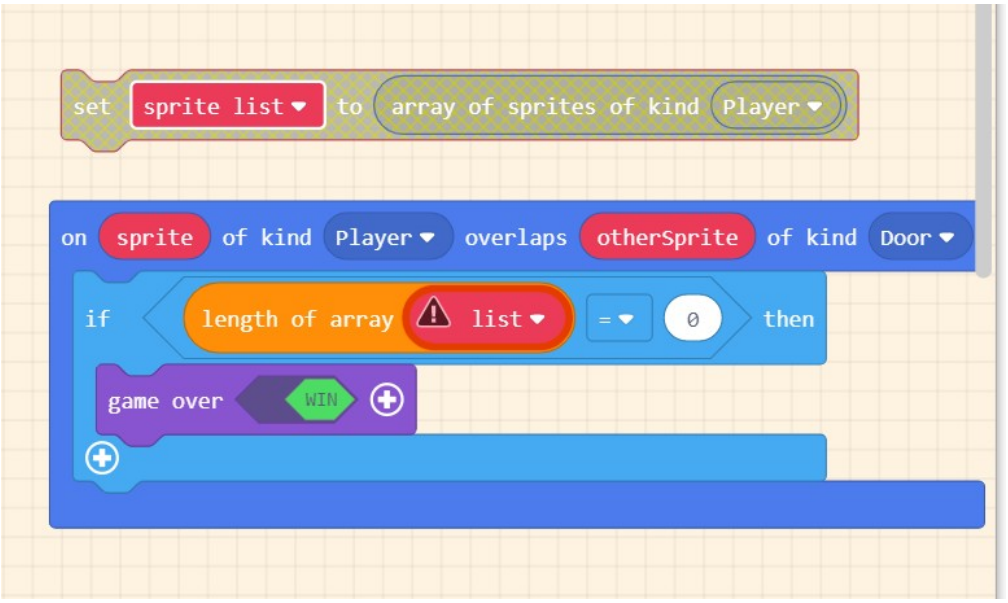
26.1.1 Add a Logic Step into our Overlap Listener

To add this patterns we need to add some logic the overlap listener which checks to see if the player is touching the end goal. Drag in an **if then** logic block and follow up with a **0 = 0** operator comparison block. From **Advanced > Arrays** drag a **length of array list** block into the first side of

the comparison block.

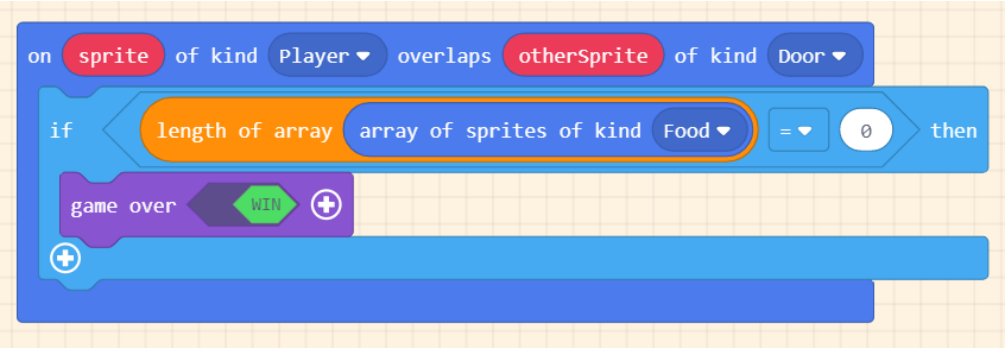
26.1.2 Check the number of Food Items

Check the number of food items by replacing the list variable with a real list. In this case the list of food items. To do this drag in from **Advanced > Arrays** a block that says **set sprite list to array of kind Player**. Drag this into a blank area near our overlap listener as we only need on part of it.



collect all food

So Drag the blue block saying **array of kind player** inside the **length of array** block, and change Player to Food. Check with the screenshot below.



collect all food

26.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example, do you have to collect all food or can you progress without doing that?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

27 Collect Points

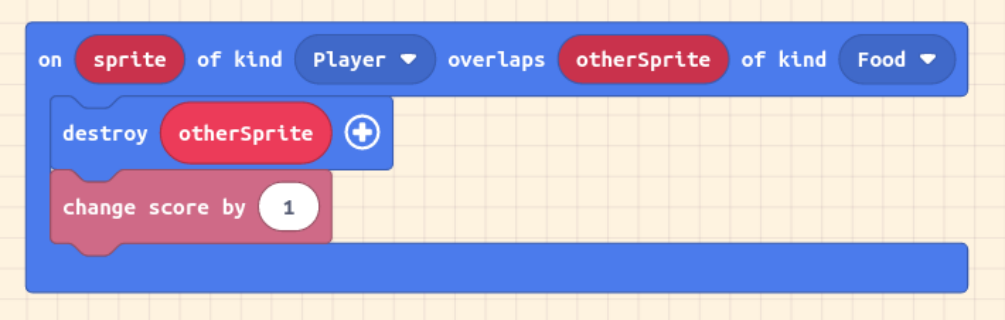
collect points

- **Name:** Collect Points
- **Description:** The player is able to increase their score by collecting food or stars or other tokens.
- **Need for Pattern:** Having the ability to **collect points** is a way of increasing player motivation and adding depth to a platformer level. For example when a player is replaying a game they may try to maximise the number of points they collect rather than just having a goal to progress as far as possible in levels.
- **Related Game Patterns:** Add Lives [related], keyAndDoor [related]
- **Coding Concepts involved:** Data, Events
- **Links to other Computing Patterns:** , Change Listener, Systems Dynamics

27.1 How to implement this Pattern in MakeCode

27.1.1 Step by Step instructions

Collecting points so that you get one point every time you collect some food (or similar) involves adding the **change score by block** to the overlap listener block.



collect points

27.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. For example, does your point score only increase by one (or your chosen number) as you progress

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

Some next steps you might want to add player lives or some kind of enemy if you haven't already.

28 Power Up Speed

power up higher speed

- **Name:** Power Up Player Speed
- **Description:** The player is able to move faster if they collect a token that acts as a way of powering up their abilities.
- **Need for Pattern:** Having a **Power Up Faster Player Speed** is a way of increasing the interest and challenge of our game. The designer is able to create time or enemy related challenges that the player can only beat if they collect the power up.
- **Related Game Patterns:** Add Timer [required], Following Enemies [related], Change Shape of Levels[related]
- **Coding Concepts involved:** Data, Loops, Events
- **Links to other Computing Patterns:** , Change Listener, Input Event, Systems Dynamics

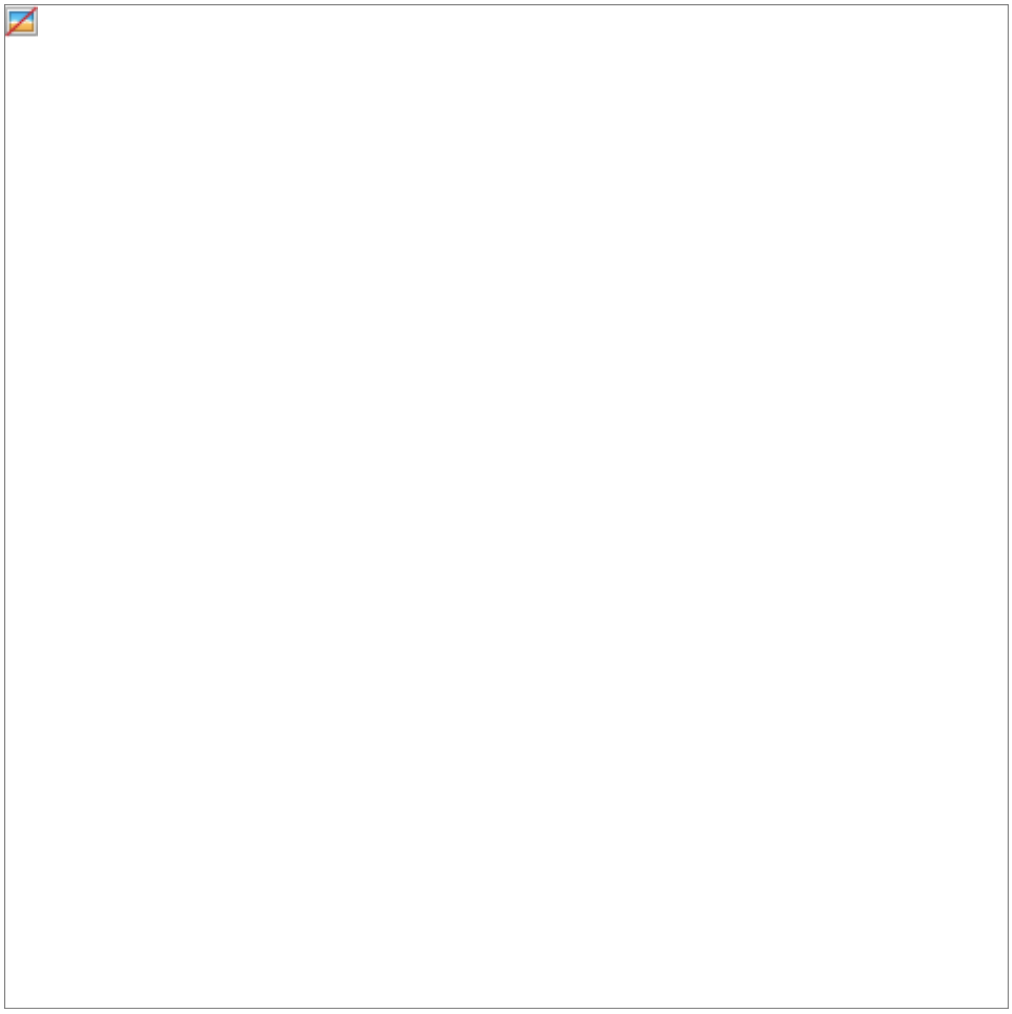
28.1 How to implement this Pattern in MakeCode

28.1.1 Requirements

This tutorial assumes you have a timer on your platformer.

28.1.2 Create our power up sprite

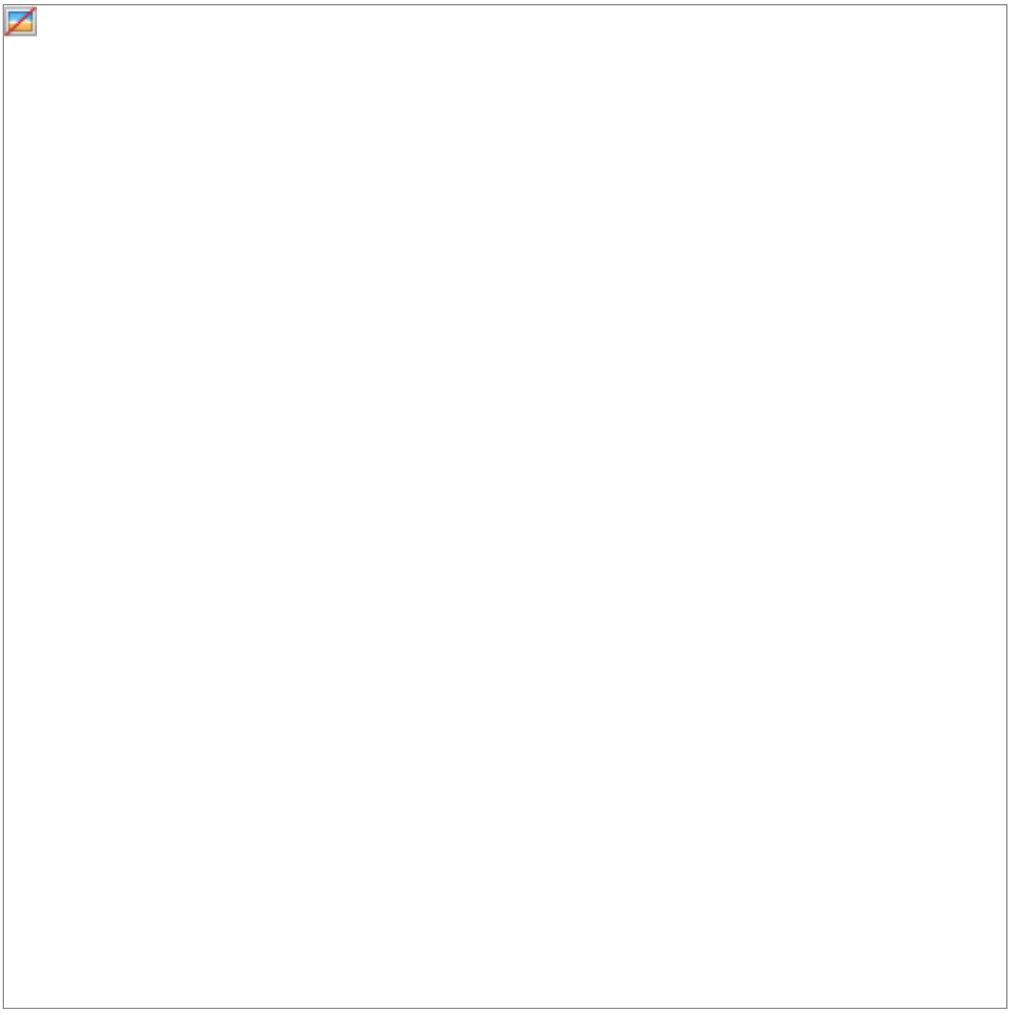
To create a power up we will replicate the for element loop used to display the Food elements on our game. So the first step is to duplicate that and to make changes to the values. You can find more detailed instructions on how to do this is the create static enemeies pattern. To start change your level Tilemap design to add a new colour tile to act as our power up and change the value of your timer to be one the player would not be able beat if they travelled at normal speed.



power up speed

In the example above you can see that the layout of the level is longer than the starting one. This is due to this pattern working well will longer levels.

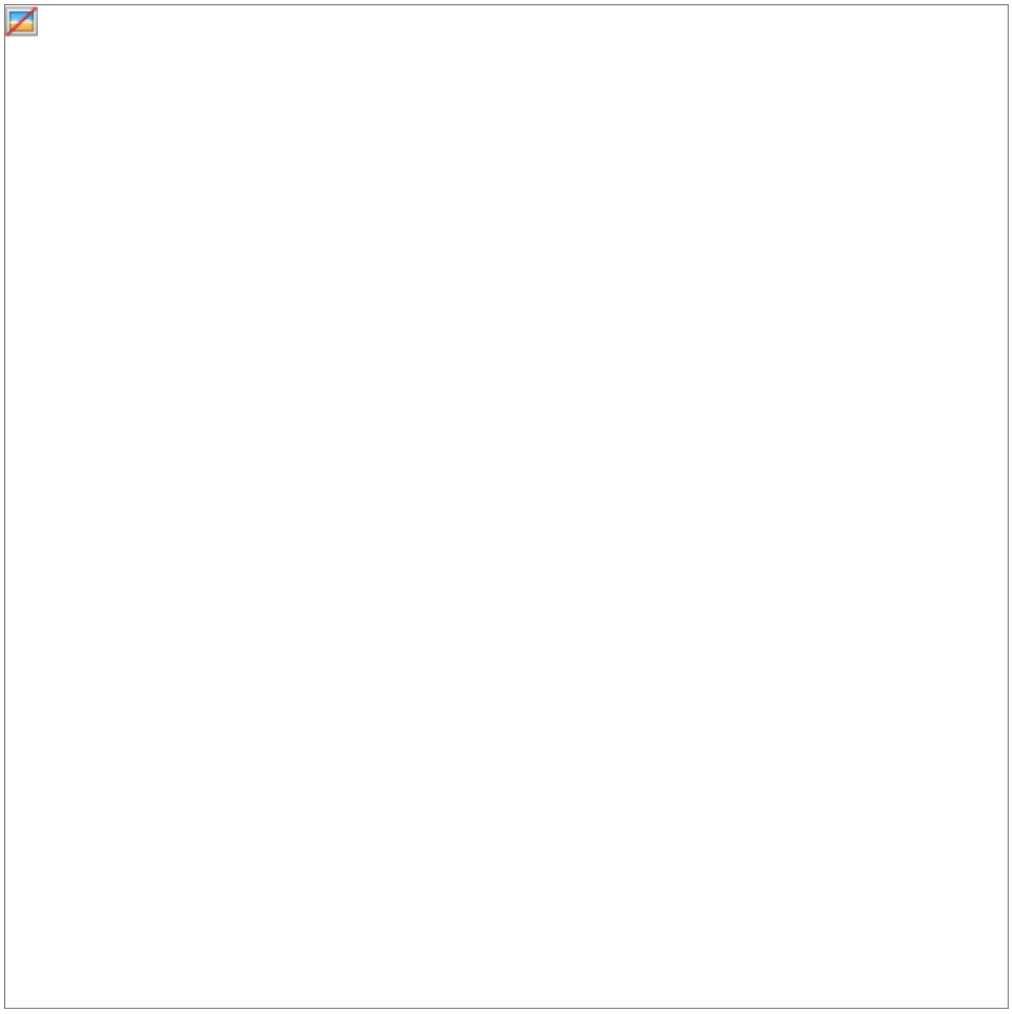
Follow the code pattern shown in the screenshot below to make the power up appear in our

game. 

28.1.3 Create a new Variable

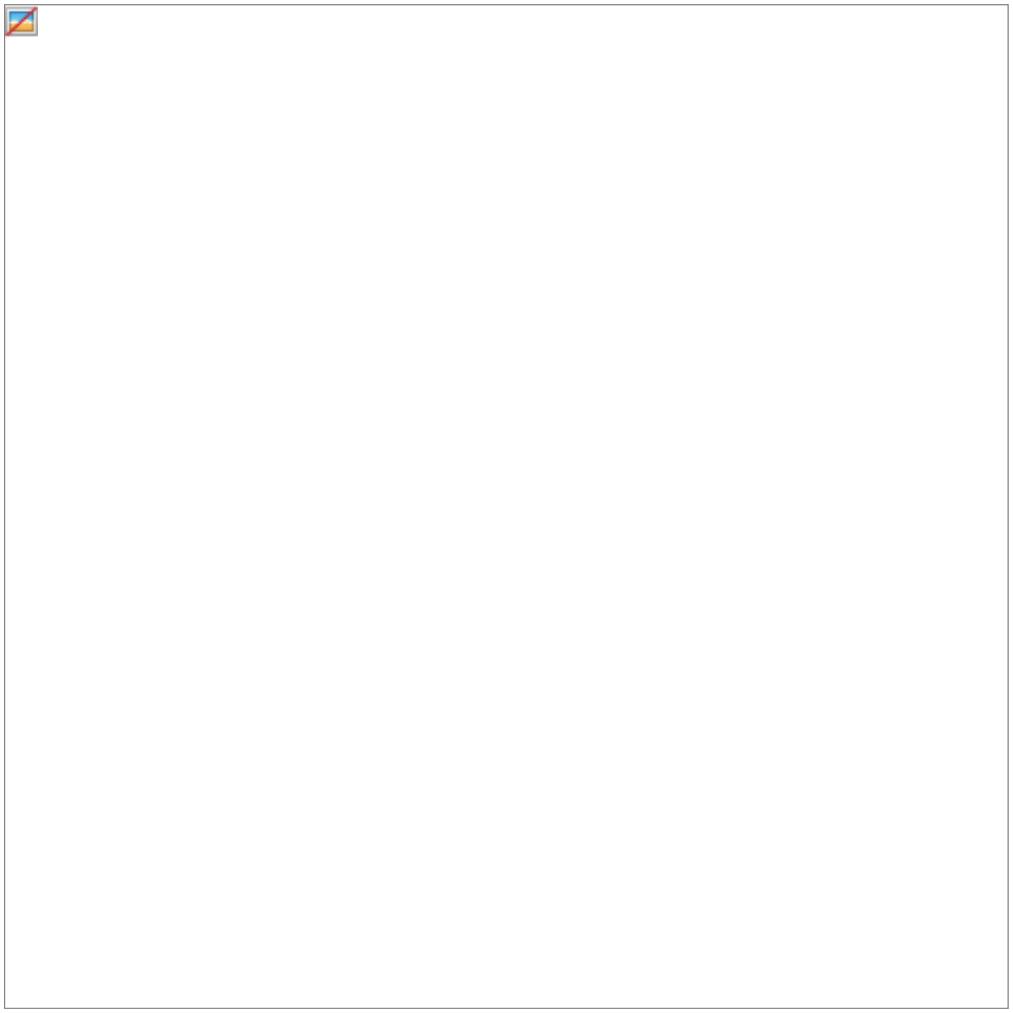
To be able to change the speed of our player we will need to create a new variable for our game. This will then be used when the player presses left and right buttons. To do this create a new variable called playerSpeedVelocity and and set it to the vx value you have in your **move sprite with buttons** block.

You can now replace the original velocity x (vx) value with this new variable.



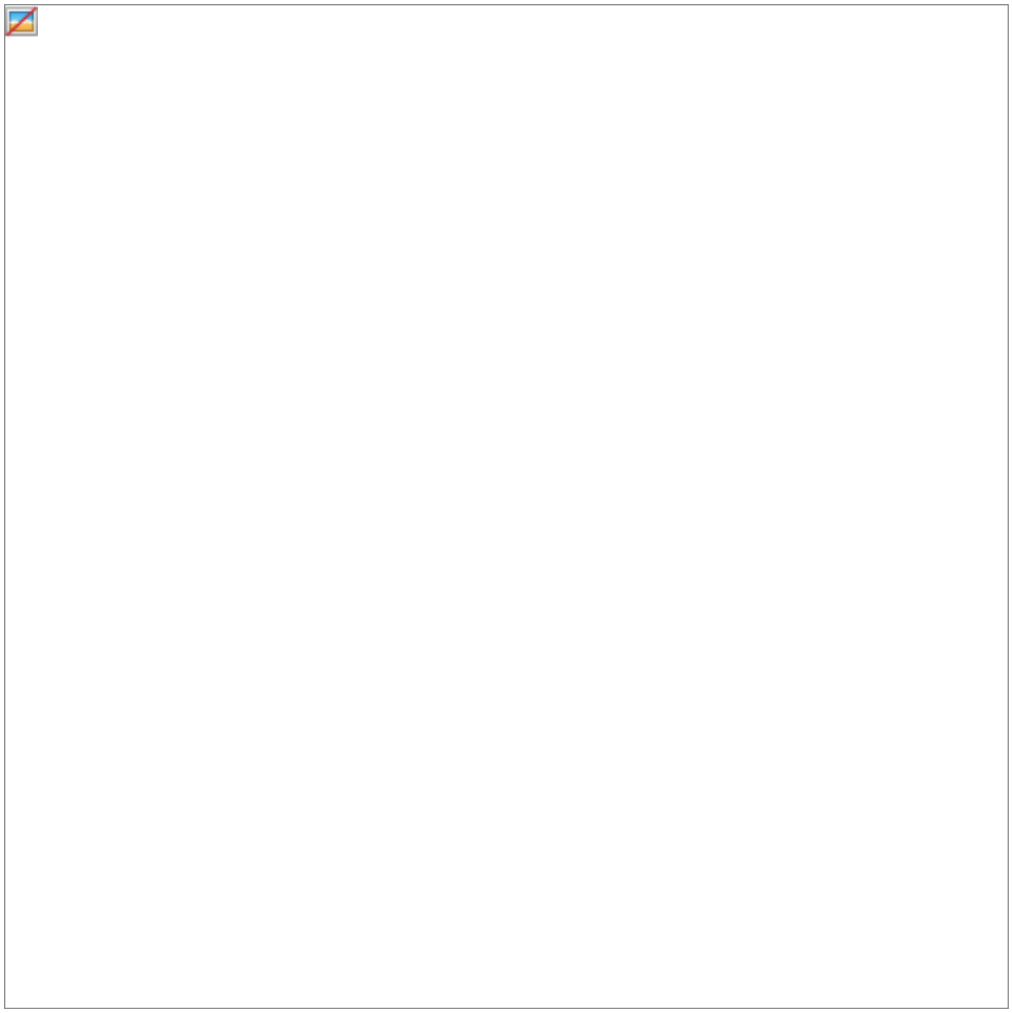
28.1.4 Create the new Variable Value

Then we will add a condition listener to see when the player is overlapping this power up. When they do we can make the power up dissappear and increase the velocity of the players speed. By increasing the value of the variable that we have just created.



28.1.5 Move Input Controls into a forever block

We need to do one last thing to make it possible for our player speed settings to be updated as we are playing the game. We need to move the block setting the player speed out of the on start block into a loop that is constantly being updated. If not that setting is set at the start but never updated.



power up speed

28.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects. To check that you are making the most of this pattern you can ask yourself the following questions:

- Can you complete the level only if you after you collect your power up?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.

As a follow up to this maybe you can make changes to the timer for different levels.

29 Remixing a Platformer Game - Random Doubling Enemies

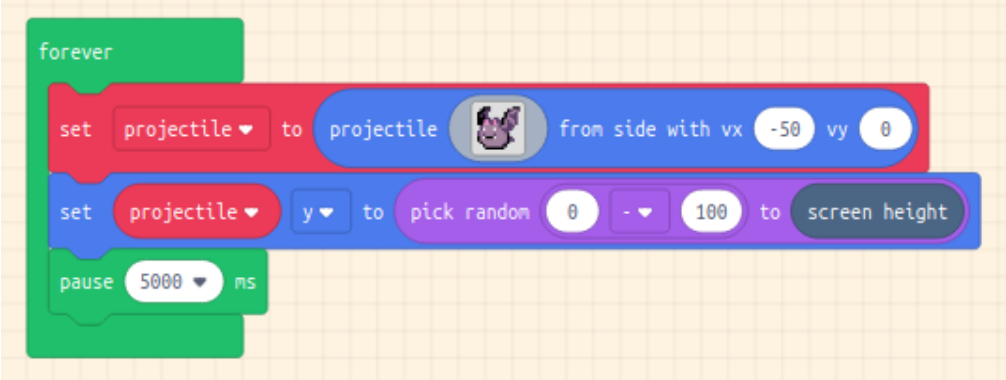
random doubling enemies

- **Name:** Randomly spawning enemies that double in number
- **Description:** This pattern adds enemies to our game that appear in random places. The number of enemies on the screen also increases as the game progresses.
- **Need for Pattern:** Having a **randomly spawning enemies that double in number** is a way of increasing the Challenge in our game. It is also an example of a feedback loop.
- **Related Game Patterns:** Add Static Enemy [related],
- **Coding Concepts involved:** Data, Loops, Events
- **Links to other Computing Patterns:** Systems Dynamics, Reinforcing Feedback Loops

29.1 How to implement this Pattern in MakeCode

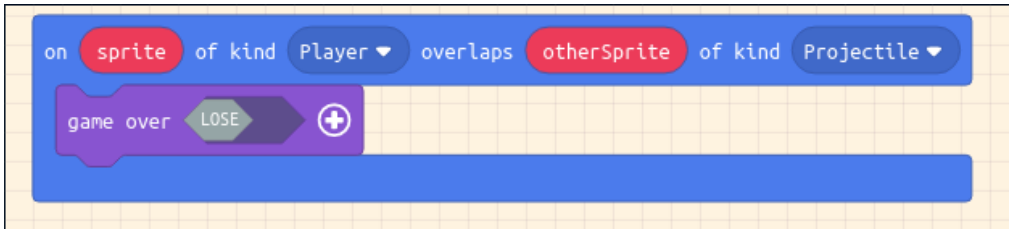
29.1.1 Create our Random Enemy

We will create our randomly appearing enemy by creating a forever loop which creates an enemy of a type projectile that starts at the side of the screen and moves left. So the enemy doesn't always start in the same place we will make the height value (y) random. See the screen shot below. Add in a pause to the forever loop so it this new projectile happens every 5 seconds to start with.



random doubling enemies

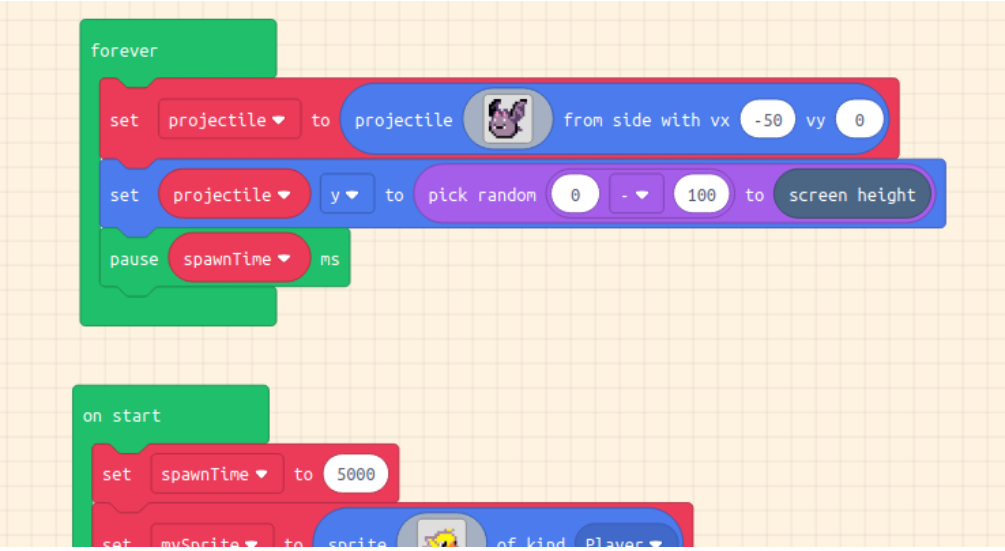
We now code a listener event that creates game over if there is an overlap.



random doubling enemies

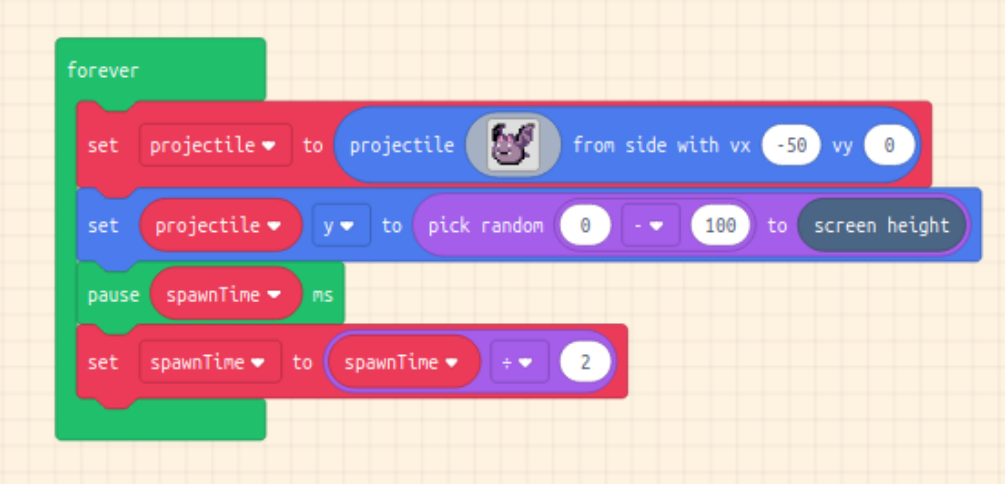
29.1.2 Increase Spawn Time

A common pattern is to increase the spawn time of our enemies to increase the challenge. To do this we will first create a new variable at the start of our game called spawnTime. Replace our value in the previous fover loop with this variable.



random doubling enemies

We now need to start to decrease this spawnTime variable. One easy way of doing that is to half the value every time the event runs by dividing it by two. This creates a reinforcing feedback loop so things might get crowded with projectiles very soon.



random doubling enemies

Your player will have to be fast.

29.2 Test your Changes and Next Steps

Test your game to check that your changes have the desired behaviour and that there are no side effects.

To check that you are making the most of this pattern you can ask yourself the following

questions:

- Does the number of enemies increase at too fast a rate? If so how can you change the rate do it doesn’t double?
- Would you want to have a maximum number of enemies on the screen at once or a maximum rate of increase to avoid?

This Game Pattern is one of many allowing you to make improvements to your platform game and to learn coding and wider computing concepts. Find more on the Game Pattern page.