# Design Stage

## Decomposition

In this section of the design stage, decomposition will be used to break down the problem and identify certain aspects of the game that need to be completed and then put together to obtain the finishing product. This simplifies the game’s next development stage (implementation) as I will have a clear list of many very small problems that I need to complete, after which I can simply put the solutions together to finish up the game.

# A diagram of a game AI-generated content may be incorrect.

I have made in depth diagrams for the mainly problematic branches above to plan out what each branch might need and what must be considered and understood in order to be implemented into the game successfully.

Starting off with the menu, a part of the Games extras branch as shown above:

# A white board with writing on it AI-generated content may be incorrect.

Menu – the piece of code that will make the functioning and draw the menu onto the player’s screen.

* Pause – a feature of the menu, if the menu is open, the game is paused so the player can freely access the menu or adjust the settings without the game continuing. This is important because if the player is busy adjusting settings for example, it wouldn’t make sense for the game to continue playing in the background. This is because if the player is in the menu, they are therefore not currently playing the game, so they are wasting processing power, and secondly, the player’s data may be updated as the game is running while they are trying to use the menu, which would be unfair on the player if for example, they lost a life while they are trying to use the menu.
  + Un pause – when menu is closed, the game is un paused because the player has gone back to the “currently playing the game” state, as they do not need the menu anymore.
* Controls guide / manager – contains information on controls for the game
  + Controls list – provides a list of controls the player can use in case they forget or don’t know what different key inputs do
  + Controls explanation – next to each control in the control list, there will be a small description that describes to the player what each control does and perhaps some things to note for the player when using the controls. This is so the player can think about how to make the best use of each control, for example, jumping to get from one platform to another.
* Menu screen design
  + Buttons – clickable / interactable areas that perform a function, such as opening the settings or closing the menu. This is so that players who are more comfortable with pressing buttons to access parts of the game will be able to do so. The menu will be openable both via a keyboard input, or by clicking a button, this provides variety, and in case the player forgets the keyboard shortcut, they can still open the menu anyway without issue.
    - Settings – this is one of the buttons, clicking on this button will allow the player access the settings, so they can adjust parts of the game if they want to.
    - Close menu – this allows the player to close the menu if they forget the keyboard shortcut to close the menu.
    - Open/close control guide/manager – while inside the menu, the player will be able to open up the screen that has all the information about the controls.

* + Background screen – as I am not sure how to literally make a menu that is opened, the idea is to essentially cover the entirety of the player’s game screen with a background, then draw and add the menu’s functionality and objects onto the background screen, which essentially creates the illusion of a menu, except it actually is a menu.
  + Menu title – this is drawn on most likely on the centre towards the top of the screen, to indicate to the player that they are in the menu. If this wasn’t here and for example the player accidentally opens the menu, they might not know they are in the menu and think the game has been bugged and will not know how to return to the main game.

The second expanded diagram I have included is for the controls and mechanics of the game:

# A white board with writing on it AI-generated content may be incorrect.

Mechanics and Controls – the structure and necessities needed for the pieces of code that manage and work on the mechanics and controls of the game.

* Base mechanics – mechanics that are available to the player by default and will always be there, this is needed as the player wouldn’t be able to play or interact with the game if there is nothing for the player to do.
  + Movement – a very fundamental piece of the game is being able to move as it is a platformer game, if the player can’t move then they aren’t really playing the game.
    - Keyboard key “A” input – pressing the “A” key on the player’s input device (keyboard for this game) allows the player to move to the left.
    - Keyboard key “D” input – pressing the “D” key on the player’s input device allows the player to move to the right.
    - Keyboard key “Spacebar” input – pressing the spacebar allows the player to perform a jump, gravity is applied as soon as they start jumping so they fall back down, and they can not jump again until they are standing on a stand able platform. This is also important as the game does not have a freely omni directional movement system, so they cannot just fly around the map, the jump mechanic makes the player’s movement feel more realistic as a character in a platformer game ideally should not be able to fly, as this would defeat the purpose of completing the parkour and obstacle courses in the game. This also means there is no key input that allows the player to go down, as the player realistically isn’t able to manipulate their gravity.
  + User interface “UI” – on screen mechanics that allow the user to interact with certain parts of the game.
    - Open and close menu button – allows the player to access the menu even if they don’t know the keyboard short cut to open the menu, by clicking on the button
      * Menu buttons – allows player to interact with menu, explained under menu’s structure diagram
* Interaction mechanics and functionality – the games response to the player’s actions, this is so that when the player performs an action, that action is actually carried out and does something in game that provides a result, for example if the player moves into a block, they won’t be able to walk right through the block as they would be colliding with it.
  + Movement collisions – broken down into two separate directional collisions. Movement collisions manage and monitor how the player’s character is interacting with the game, for example if they stand on a platform, it ensures they don’t fall through by continuously resetting the player’s position to the boundary of the platform’s side the player is on or at.
    - Vertical collisions – this set of instructions /code will ensure the player does not fall through a platform they are standing on or falling onto, and ensures they cannot jump through a solid platform from below.
      * Jump collisions – when jumping, ensures player don’t jump up and through a solid platform (bottom collisions)
      * Falling collisions – when falling, ensures player doesn’t fall through from top to bottom of a solid object (top collisions)
    - Horizontal collisions – when the player is moving sideways, if they hit a block, they do not go through it from the side they hit it from. i.e if they walk from the right, to the right side of a solid object, they do not start walking into and through to the other side of the solid object via the solid object’s left side, and if they are walking from the left to the left side of a solid object, they don’t walk into then through the solid object from its right side to the left.
      * Moving left – when moving to the left (the player), if they move into a solid block, they don’t go through (right collisions)
      * Moving right – when moving to the right, if the player hits a solid object, they don’t go through the object (left collisions)
  + Timer pause – when the player goes into menu for example, the game’s timer is paused because it wouldn’t be fair on the player to lose their time remaining to complete the level when they are trying to for example, adjust a setting in the menu.
  + Stop movement and drawing on pause – when the player opens menu the player’s positional data and general data is frozen so they cant for example, somehow lose a life while the game is paused, or lose score. (maybe be able to reduce load on CPU?)

The next expanded diagram describes and lays out how the player’s class and systems will be made and implemented.

A whiteboard with writing on it

AI-generated content may be incorrect.

Player – the class that the player’s character will be made from. This is what the user will use to play the game, they will control this player.

* Attributes – this part of players contains all the information about the player’s character. This is so that the player’s character can function and interact with the game properly, and it also has extra information that the player may want to view to help them complete the game. Most of these attributes’ are encapsulated as the player does not need to know for example, the exact size their hitbox.
  + Health – this is fundamental to the player’s character, as it will be used to operate on the lives system and loss/defeat system for the game. It also helps the player learn to be cautious with what they do and think thoroughly before acting as to make sure they don’t make irrational decisions that may hinder them from completing the game. If it reaches 0 then the player loses a life and they’re progress is reset to the last checkpoint.
  + Lives – this is a counter of how many attempts remaining the player has to beat the level / game. It helps provide a challenge to the game’s simplistic genre. If this reaches 0 the player loses the game.
  + Standing on ground – this attribute is a Boolean value that is used to determine if the player will be able to jump or not, it wouldn’t make sense for the player to be able to continuously jump as much as they want while in the air because that would basically be allowing the player to fly, defeating the purpose of completing the game.
  + Players hitbox – the attributes underneath this set is used to check for and deal with player collisions with objects.
    - X position – the horizontal position of the player, the value is a part of the top left corner of the player’s hitbox.
    - Y position – the vertical position of the player, part of the top left corner of the player’s hitbox.
    - X velocity – private attribute that is used to update the player’s position, meaning if they are moving left or right, the x velocity will have a value which will be added on or subtracted from the x position to draw and show that the player has moved horizontally in a direction.
    - Y velocity – private attribute that is added on to the player’s y position, this enables the player to jump whenever they press the jump control, and fall whenever they fall from a platform, or after jumping.
* Drawing of player / picture – this is one of the most important parts of the player as if the player isn’t drawn on, then the user does not know what is going on in the game because they don’t know if their inputs are doing anything. Drawing the player shows the user the results of their actions.
  + Image file – this may be added on top of the player’s drawing to give a more detailed and interesting appearance to the game, as drawing with pygame’s simplistic shapes may be boring to look at.
* Statistics – extra information about the player that isn’t necessarily used to control and get the player to interact with the game’s systems, but may be important for the user’s experience. Some stats may be displayed in the menu as they aren’t relevant to the majority of the gameplay, for example wins and losses, the player doesn’t really need that information in order to beat the game, they are just information.
  + Score – gives the player an idea of how well they are doing, so they know if they are close to beating the level they are on, or if they need to change up their strategy.
    - Score incrementor – increases the player’s score value every time they perform an action that increases score (i.e gain a collectible).
    - Score decrementor – decreases the player's score value every time they perform an action that decreases score (i.e lose a life).
    - Score counter – displays the score value of the player as a number to the player on the drawing screen. This lets the player see how well they are doing. It is updated every frame cycle and after the player’s attributes have been updated, as the player has to gain the score, before the score can be updated as updating the score before the player gains any provides out of date information to the player.
  + Wins – number of times the player has won the game.
    - Every time the player’s wins is increased, a background will be drawn on to the screen with drawing to indicate to the player that they beat the current level or game.
  + Losses – number of times the player has lost the game.
    - Every time the player loses all their lives (a loss), the player will be presented with a screen that tells them they lost, so they know they have to restart from the beginning.
  + Levels – Container with information on all the playable levels in the game.
    - Existing levels – all levels that can be played in the game, so the player can see how long the game is. A list.
    - Unlocked levels – levels the player can access as they have already beaten the previous levels up to the unlocked levels. This will provide the players with more opportunities to play different parts of the game as they progress. A list.
    - Locked levels – levels that the player has not yet reached up to. This is to prevent players from skipping to hard levels that may need knowledge or mechanics that is unlocked in previous levels that they have not completed yet. A list.
  + Collectibles – items / achievements the player gets for playing the game, some of which will be beneficial to the player in completing the next levels they play.
    - Unlocked collectibles / obtained – list / collection of items they have found and not yet used up. A list.
    - Locked collectibles / missing – collection of items they don’t yet have. This can help the player in completing the game as it also gives them a side objective of looking for items they haven’t obtained, in case they already finished the main game and want to keep playing. A list.
  + Lives used on current level – extra information so the player can keep track of how well they are doing, it might be important for developing decision making skills as they can use it to expect how hard the next levels will be and allows them to choose if they should develop skills before moving onto the harder levels. A counter.
    - Lives remaining – the total number of lives they have left, may use a get method from player class to obtain the value for the number of lives, this will then display that number. A counter.
    - Lives regained from current level – a number that is out of another number (i.e 5/10), this can be used by the player to see if they can gain any more lives before moving on to the next levels, helping them ensure they don’t lose the game. A counter.
  + Miscellaneous – extra pieces of information that the player might want to see /achievements
    - Time played – tells the player how long they have been playing the game for the current session.
    - Levels attempted – how many times the player has attempted each level they have played.
    - Levels high scores – the highest score the player has obtained on each level; a report of how well they did on their best attempts on each level.
    - Time spent on current level – displays how much time the player has spent trying to beat the current level, could be useful for the player to decide if they need to go back to improve at certain skills before reattempting the level they are currently trying to move on from.
* Controls – controls that the player can perform, explained in detail in the control and mechanics section
* Singe-player game mode – the game is a one player game, so I don’t need to worry about multiple players at a time, meaning the game can be simply implemented and played as long as the player has the game file, therefore don’t need to set up a multiplayer server side.

## Usability Features

Background drawn on screen to give the illusion of a menu

Title for menu to indicate player is in menu state

Menu example design

A menu written on a lined paper

AI-generated content may be incorrect.

Button to open settings

Everything is placed near the centre so the player’s attention is drawn to the important bits

Button to view list of controls along with their descriptions

Buton to close menu in case player doesn’t know the keyboard shortcut to close menu

Button to close settings if player doesn’t know the keyboard shortcut to do so

Settings title

Screen drawn on top of menu screen to give illusion of a settings menu having been opened

Example setting options that the player can change by an integer value input

# A whiteboard with writing on it AI-generated content may be incorrect.

Drop down setting options may make it easier for players to adjust certain settings

Input boxes that allow for change of settings

Being able to scroll down the settings allows for more setting options to be added without sacrificing the font size of each option so the player can adjust many settings as well as read them easier without needing to zoom in or look closely

Control descriptions and controls take up the majority of the screen so that as many of them can be fit onto the screen and also draws the player’s attention so they can easily see what the controls are

Similar to settings sub menu, a background is again drawn onto the current screen overlapping the menu screen to show controls section

# A piece of paper with writing on it AI-generated content may be incorrect.

Ability to scroll down allows more controls to be added, if there will be that many controls

Closes controls menu in case the player doesn’t know the keyboard shortcut to close it

Announcing to the player in capital letters places emphasis so the player knows they won

Menu button

# A white board with writing on it AI-generated content may be incorrect.

Buttons that provide the player a direction on what to do now that they have finished the level they are on or the game. If game is completed the “next level” button won’t appear

Placeholder, but a drawing that emphasises a win will help improve the atmosphere to make it feel more accomplishing

All that is on the loss screen will be essentially identical to the win screen to provide the same magnitude of effect, but the opposite way

A paper with writing on it

AI-generated content may be incorrect.

There is no “next level” button as the player has not beaten the current level. (if this level was beaten previously, maybe it could be an exception to let “next level button to exist on this loss screen as well)

Game title in a special font and capitalised so the player knows what game they are playing and that they are in the main menu screen

A paper with writing on it

AI-generated content may be incorrect.

Like with most of the other menu screens, buttons are in the centre to draw player’s attention so they know what choices they can do in the game (play from beginning or go to menu or select which level to play from)

## 

A small rectangle or screen drawn onto the main screen as an inventory, this enables the player to see the timer and other stats of the player while they are accessing the inventory. I am going to implement it like this because when the player is accessing the inventory, unlike accessing the menu (menu state), the player is still playing the game (game state), so completely covering up the rest of the game would be unnecessary and also unreasonable as the player may want to see how much score or time he has. For example, he might need to see his health, then decide what type of item in his inventory he needs to use.

## A graph on a white board AI-generated content may be incorrect.

Title indicates clearly to player they are in inventory

Player might have many items and not all of them can be viewed within the small screen, scrolling will enable them to have more items

Placeholder / example grid, items /collectibles will be stored in these slots, player can press on a slot to access / use the item

All these statistics and attributes are at the top of the screen (maybe to the side) because they important so they need to be seen by player, but also don’t need to be blocking the player’s field of view

A graph on a piece of paper

AI-generated content may be incorrect.

Player’s image

Menu button to access menu in case player doesn’t know the keyboard shortcut to open menu

Player’s rectangle which is their hitbox and used to detect collisions

Level selector menu, again, like all other menus the title is in the centre near the top to draw the player’s attention and tell them clearly where they are in the game

A white board with lines and text

AI-generated content may be incorrect.

Buttons that allow the player to access the corresponding level on click

If there are a lot of levels that cant fit on the screen, scroller will be placed here to indicate to the player they can scroll down in this menu to see more existing levels

## Classes

In order to keep my game organised, I will make my classes in a separate file from my main game, and import it as a module to be used in my main game. This way when trying to make additions to the game, such as levels, I can very easily make the levels simply by just creating instances of a class and then putting them into a level’s group then draw them in.

Starting off with making the levels, I will need obstacles, therefore breaking down the obstacles I will have, I will need:

* Moving obstacles
* Static obstacles
* Obstacles that deal damage (Killblock)
* Block that is a win condition (touching it beats / completes the level)

I am choosing to make separate classes for each obstacle because this significantly simplifies building the game’s levels when I come to that part. This is because by making the obstacles classes, every time I want to add a platform to a level, I can go to the level’s code, create a new instance of the type of platform, then add it to that level’s group of objects. This will help save time as I am essentially making myself a map builder. If there is a bug with a type of platform, it also makes it easier for me to debug as all the code is centralised in one spot, underneath the class’s definition.

Moving obstacles class diagram:

A list of exercises on a piece of paper

AI-generated content may be incorrect.

Static obstacles class diagram:

A close-up of a note

AI-generated content may be incorrect.

Killblock class diagram:

A close-up of a list

AI-generated content may be incorrect.

Win condition class diagram:

A list of school tasks

AI-generated content may be incorrect.

Another class I plan to use is for the player’s character. It will include all the player’s attributes (via constructor) and features (via methods). This makes drawing the player into the game easy as just like the obstacles, I create an instance of the player class and simply draw them in.

Player class diagram:

A list of tasks on a white board

AI-generated content may be incorrect.

As I will be making multiple screen such as menus and settings, I will need buttons that the player can click to interact with the screens. Therefore I will make a class for buttons as well so that every time I need to add a button I can do so quickly and efficiently.

Button class diagram:

A piece of paper with writing on it

AI-generated content may be incorrect.

# Algorithms

Below I will show all the algorithms that I am sure on how I can implement into the game.

Base movement algorithms:

* Moving to the left
* Moving to the right
* Jumping
* Falling

Settings options:

* Adjusting volume
* Adjusting display size

Camera:

* Horizontal camera scroll
* Vertical camera scroll

Level selection menu

Game on startup

Main menu

Collisions:

* Horizontal collisions
* Vertical collisions

Moving right:

A diagram of a play class

AI-generated content may be incorrect.

Moving left:

A diagram on a piece of paper

AI-generated content may be incorrect.

Jumping:

A graph paper with writing on it

AI-generated content may be incorrect.

Falling / gravity being added on:

A diagram on a piece of paper

AI-generated content may be incorrect.

Collisions:

Horizontal collisions:

A diagram of a program

AI-generated content may be incorrect.

Vertical collisions:

A diagram on a piece of paper

AI-generated content may be incorrect.

Camera scrolling :

Horizontal scrolling :

A diagram on a piece of paper

AI-generated content may be incorrect.

Vertical scrolling :

A diagram on a piece of paper

AI-generated content may be incorrect.

Game upon being opened:

A diagram of a person meal state

AI-generated content may be incorrect.

Main menu state :

A diagram on a piece of paper

AI-generated content may be incorrect.

Settings option for volume :

A diagram of a diagram on a piece of paper

AI-generated content may be incorrect.

Settings for display resolution:

A diagram with writing on a piece of paper

AI-generated content may be incorrect.

Level selection menu:

A diagram on a whiteboard

AI-generated content may be incorrect.

## Main variables and data structures

Here I will list all the key variables and data structures that I plan to and know will definitely be in the game during implementation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classification | name | Data type | Function | Justification |
| Variable | size | list | This variable is used to tell the program how large to draw the game screen | I need this variable because of the settings feature I will add, which if the player choses to change the resolution, then when they do so, this variable’s values will be changed so that the screen can be drawn to the size of their liking |
| Procedure | play | - | This procedure will run and load in the game, so after the player has pressed play in the main menu, this procedure starts and begins drawing in the game | I have a procedure for playing the game because I will have different game states such as when the player is in the menu or in the settings. This procedure ensures that the game only runs when the player is actually trying to play the game |
| Procedure | mainmenu | - | This draws on the main menu onto the player’s screen. This is run as soon as the game is opened. | This like the procedure above isolates the game states making it also easier to work on different parts of the game. Also when the game starts to be played, it allows the game to stop running this procedure and run the play procedure instead to play the game, effectively stopping the main menu from appearing while playing. |
| Variable | cameraXoffset | Real | This variable is subtracted from all objects in the game’s x positions, so that the entire map is drawn such that the player is in the centre, producing the illusion of a camera like effect whenever the player moves | This is needed because without a camera, the player would only be able to see a limited part of the entire map, as their screen will only capture as much as their display resolution. Using this variable in junction with the below one creates an effect of a camera, this allows me to add many platforms and make larger level designs and more complex ones for the player. |
| Variable | cameraYoffset | Real | Subtracted from all objects in the game’s y position so that entire map is drawn relative to the player’s y position. Part of the above variable to create a camera that follows the player. | Reason directly above^ |
| Variable | Collisionslist | List | This is a list that holds all the sprites that the player’s character collides with. | This list is needed as it will be searched into for objects that have collided with the player. This can then be used to work out both horizontal and vertical collisions. Without collisions, the game wouldn’t be a platformer as you would just fall or walk through platforms |
| Game state | Main menu | - | The screen that is seen when the game is opened | This is where the code that defines the main menu will be written in the main game file (maintains organisation of code) |
| Game state | Playing | - | The actual game itself | The code for the main game will be under this section in the main game file |
| Game state | Settings | - | Settings sub menu that allows player to adjust parts of the game | The code for settings will be under this section in the main game file |
| Game state | Level selector | - | This part allows players to select what level they want to play on | This also maintains organisation of code by keeping blocks of code that are on level selection under this section |
| Procedure | Settingsopen | - | opens up the settings menu so the player can adjust things | All code that gives the settings functionality will be in here. Allows the settings to be easily opened or closed. |
| procedure | Levelselector | - | Opens up the menu that lets the player select a level | Code that manages the level selection menu is kept here, this allows this sub menu to easily be opened or closed. |
| Variable | Done | Boolean | Determines if the player has finished playing the game and closed it or not | This variable is checked every cycle so that the code can close the game if the player has chosen to quit it |
| Variable | Player | Instance | The player’s character in game | Used to dra w the player into the game, this allows the user to see their character and what they are doing in game. |
| Module | Colours | - | Contains all the colours for drawing I may need | allows me to easily access colours I want to use for drawing purposes in all my files. My levels will be in separate files so it saves me time from having to redefine every colour I may need to use in every single level file. |
| Module | Classes | - | Contains all the classes to be used in game | I have this as it keeps my project organised, writing all the classes separately in this file allows me to more easily identify future bugs and errors. Also because my levels are drawn in separate files, they each need to access the classes for the platform types, so it saves time as I don’t need to redefine each class in each level file. |

## 

## Test Data

Movement controls:

|  |  |
| --- | --- |
| Test data | Validation |
| “a” | Valid |
| “d” | Valid |
| “e” | Invalid |
| “f” | Invalid |
| Spacebar | Valid |
| “s” | Invalid |
| 9 | Invalid |
| [ | Invalid |

Open menu:

|  |  |
| --- | --- |
| Test data | Validation |
| “m” | Valid |
| Click on menu button | Valid |
| Click not on menu button | Invalid |
| “p” | Invalid |
| Click on edge of menu button | Boundary but valid |

Open settings:

|  |  |
| --- | --- |
| Test data | Validation |
| “n” | Valid |
| “l” | Invalid |
| Press on settings button in menu | Valid |
| Press on edge of settings button in menu | Boundary but valid |
| Press not on settings button in menu | Invalid |

Score board:

|  |  |
| --- | --- |
| Test data | Validation |
| Updated score < 0 | Invalid |
| Updated score > 0 | Valid |
| Updated score = 0 | Boundary but valid |
| Updated score is an integer | Valid |
| Updated score is not of integer data type | Invalid |

Open level selector:

|  |  |
| --- | --- |
| Test data | Validation |
| Press on level selection button in menu | Valid |
| Press on settings button in menu | Invalid |
| “e” | Invalid |
| “l” | Valid |
| “,” | Invalid |
| 2 | Invalid |