The Idea

We will make a 2D platformer game, like Mario or Hollow Knight. The aim will be to reach the end of the level without dying. Along the way the player will face enemies that will try to stop the player, as well as parkour to reach the next stage.

The game will be for PC only and written in Java. We chose Java because of its simple GUI system that is portable across any platform with Java installed

Interaction with the player

The player will be able to interact with the game with 2 main methods:

* Playing the game, this involves the movement controls, the game window and the GUI on screen.
* Through menus, these involve the start, pause and options menu.

Having a simple interface means that it will be easy for any player to be able to navigate the game menus, knowing exactly how to edit a specific setting or getting back to playing the game.

Start menu concept:

A picture containing diagram

Description automatically generated

The play button leads to a level selector where you select which level to play (levels unlock after finishing the previous level - the first level is a tutorial level that will teach the player the basic controls and, some of the enemies and the basic game concepts). The options menu will have settings like the video and audio settings, the menu will also be accessible through the pause menu.

System Architecture:

The game will have 7 main classes:

* Rigidbody: includes functions for collisions as well as data such as the position and rotation.
* DynamicRigidbody (extends Rigidbody): contains functions for movement.
* Entity (Extends DynamicRigidbody): contains information about health, etc
* Player (extends Entity): includes movement controls as well as attacking.
* Enemy (extends Entity): includes enemy damage, Enemy movement AI.

Most other classes with be small variations of these classes (e.g. enemy or platform types). The gameplay will be made up of a mainloop, this includes the updating the level by dt if the game is being played, and calling the renderer. The renderer will handle rendering the levels and the menus.

This is a basic flow chart outlining the barebones of the game loop and handeEvents subprocess. The gameloop will be run when playing a level. The mainloop will call this when a level is clicked on in the menu selector.

Diagram

Description automatically generated

Requirements

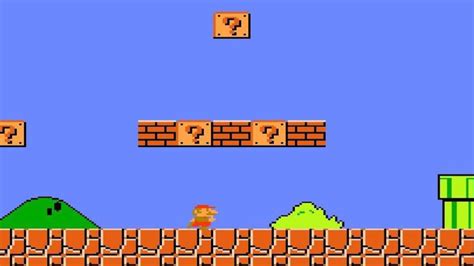
System Requirements:

* Java installed on any operating system (Swing is platform independent)
* Keyboard
* Mouse

User requirements:

* The menus must be easy to navigate.
* The controls must be simple enough that anyone can use them.
* Must have enough gameplay to keep the user entertained.
* Should be able to have the window full screen or windowed.

Other platformers:



Super Mario Bros, one of the most iconic platformer games ever. The player traverses the levels with only movement controls, killing enemies by jumping on their heads. This game will have a large influence on what our project will be like as it is a successful example of what this kind of game could feel like.

Development:

The game is split into different packages. The Game package contains all of the classes used in the game (the Player, Enemies as well as Rigidbodies), the Application package contains the classes for all of the menus, as well as levels, the Physics package contains physics classes such as rays and vectors.

Every unique level is placed within a sub-package in the Application package and inherits from the abstract class Level. The abstract class contains all of the rendering and logic that every level will use, then to setup a level, all that is needed is to create a new class that inherits from it, set up constructor and build the level manually within the initLevel procedure. This allows for the game to be easily scaled upwards and more levels to be created easily.

The menus follow a similar pattern to the levels with some slightly different logic and rendering. There are 4 main menus within the game. The main menu which is loaded at the start of the game, the pause menu which is opened when the player presses esc while playing the game, the level selector which is accessed through the main menu and which displays the levels that the user can play, and the options menu which contains all of the changeable options in the game.

Below is diagram to show how each of the menus can be accessed:

Diagram

Description automatically generated

Testing with HCI methods (Cognitive):

Target – Change an option

Steps:

[From Main menu]

1. Click on the Options button, this brings you to the options menu
2. Change the option

[From playing the game]

1. Open pause menu
2. Open Options menu
3. Change the option

Conclusion:

Changing any option is incredibly easy, it is self explanatory from the menus, the only thing might be is that the option that they are wanting to change doesn’t exist.

The collision system:

As everything in the game is a rectangle, it makes the collisions very simple. We used a ray vs rect approach to the collisions. This works by comparing a ray (the velocity of a moving rigidbody such as the player) to a rectangle (a different rigidbody).

rayVsRect:

* Finds the length along ray where the ray would collide with X and Y coordinates of the rectangle corners.
* Checks that the