**Design**

In order to create my project I am using an extensive OOP model. This encompasses the game logic with the player and player states as well as the GUI.

My player class will have an attribute for each of the states the player can be in. Each frame the update method for the player’s current state is called. This is an example of overriding where the update function behaves differently depending on the current state of the player. The player class also defines many attributes relating to movement. The player class also defines many methods, these consist of methods to check if the player is touching a wall or is touching the ground. These methods are used in state logic as well as collision detection. A method is also defined to update the player’s input handler according to the key presses on the current frame.

Each state attribute the player has inherits from a generic class named “PlayerState”. This class provides the functionality for entering and exiting states as well handling inputs and updating the player.

The “PlayerGroundedState” and “PlayerAbilityState” both inherit from “PlayerState”.

The “PlayerGroundedState” will define a new attribute that will store a vector of the player’s movement inputs on the current frame. This state will override both the input handler method and the update method.

The “PlayerAbilityState” will also define a new attribute that will store a vector of the player’s movement inputs on the current frame. A boolean flag will also be defined which stores whether the ability is done or not. The update method is overridden here.

These two states are super states which groups similar states together as they will share some logic.

Each state that the player can be in will inherit from “PlayerState” or one of the two super states. This allows for the logic and handling of the player to differ depending on the current state of the player. This is done by overriding the existing methods in each of the states. Any checks for a changes of state in the player occur in the update method.

Aggregation Diagram

Graphical user interface

Description automatically generated

Application

Description automatically generated with medium confidenceDiagram

Description automatically generated with medium confidenceA picture containing application

Description automatically generatedInheritance Diagram

Diagram

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

Class Diagram

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Inherits from “BaseEntity”

Shape

Description automatically generated with low confidence

Inherits from “BaseEntity”

A picture containing text

Description automatically generatedText

Description automatically generated with medium confidenceText

Description automatically generated with low confidenceA picture containing shape

Description automatically generatedA black screen with white text

Description automatically generated with low confidence%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22PlayerGroundedState%22%20style%3D%22swimlane%3BfontStyle%3D2%3Balign%3Dcenter%3BverticalAlign%3Dtop%3BchildLayout%3DstackLayout%3Bhorizontal%3D1%3BstartSize%3D26%3BhorizontalStack%3D0%3BresizeParent%3D1%3BresizeLast%3D0%3Bcollapsible%3D1%3BmarginBottom%3D0%3Brounded%3D0%3Bshadow%3D0%3BstrokeWidth%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22800%22%20y%3D%22600%22%20width%3D%22140%22%20height%3D%2260%22%20as%3D%22geometry%22%3E%3CmxRectangle%20x%3D%22230%22%20y%3D%22140%22%20width%3D%22160%22%20height%3D%2226%22%20as%3D%22alternateBounds%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%22move\_input%3A%20Vector2D%22%20style%3D%22text%3Balign%3Dleft%3BverticalAlign%3Dtop%3BspacingLeft%3D4%3BspacingRight%3D4%3Boverflow%3Dhidden%3Brotatable%3D0%3Bpoints%3D%5B%5B0%2C0.5%5D%2C%5B1%2C0.5%5D%5D%3BportConstraint%3Deastwest%3B%22%20vertex%3D%221%22%20parent%3D%222%22%3E%3CmxGeometry%20y%3D%2226%22%20width%3D%22140%22%20height%3D%2226%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20value%3D%22%22%20style%3D%22line%3Bhtml%3D1%3BstrokeWidth%3D1%3Balign%3Dleft%3BverticalAlign%3Dmiddle%3BspacingTop%3D-1%3BspacingLeft%3D3%3BspacingRight%3D3%3Brotatable%3D0%3BlabelPosition%3Dright%3Bpoints%3D%5B%5D%3BportConstraint%3Deastwest%3B%22%20vertex%3D%221%22%20parent%3D%222%22%3E%3CmxGeometry%20y%3D%2252%22%20width%3D%22140%22%20height%3D%228%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E

Inherits from “PlayerGroundedState”

Inherits from “PlayerState”

Inherits from “PlayerGroundedState”

Inherits from “PlayerGroundedState”

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, text, application

Description automatically generatedA picture containing text

Description automatically generatedText

Description automatically generated with medium confidenceA picture containing text

Description automatically generated

Text

Description automatically generated with low confidence

A picture containing shape

Description automatically generated

Text

Description automatically generated with medium confidence

Text

Description automatically generated

A picture containing shape

Description automatically generated

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated with low confidenceA picture containing graphical user interface

Description automatically generated

Text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generatedText

Description automatically generatedText

Description automatically generated

A picture containing shape

Description automatically generated

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated with medium confidence

A black rectangle with white text

Description automatically generated with low confidence

Text

Description automatically generatedText

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated with low confidence

A black screen with white text

Description automatically generated with low confidence

Shape

Description automatically generated with low confidence

A picture containing shape

Description automatically generatedText

Description automatically generated with medium confidence

A picture containing shape

Description automatically generatedText

Description automatically generated with low confidenceA picture containing shape

Description automatically generated

Algorithms

Linear Interpolation: This algorithm will interpolate between two values using an interpolant. This is used to find a point some fraction of an amount along a line between two endpoints.

*function* lerp(a, b, lerp\_amount) *do*

*return* ((a \* (1 - lerp\_amount)) + (b \* lerp\_amount))

*end*

This algorithm is used in the “InAirState”, this function is called with parameters a: (playerXVelocity), b: (playerTargetXVelcoity), lerp\_amount: (playerWallJumpLerpAmount). This limits the speed the player accelerates to after executing a wall jump. This means the player can’t accelerate back towards the wall and execute another wall jump in order to climb up it.

Sign: This algorithm returns the sign of an input. If the input is greater than or equal to 0 then 1 is returned. Otherwise, zero is returned.

*function* sign(flt) *do*

*if* flt >= 0 *do*

*return* 1

*end*

*else* *do* *return*

*return* -1

*end*

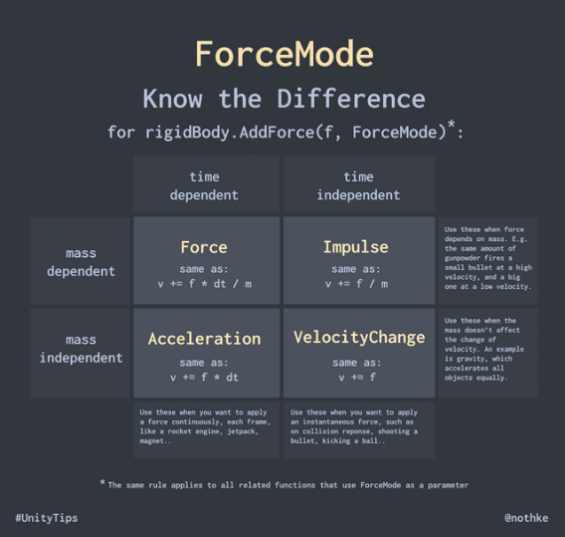
*end*

RigidBody2D.AddForce: This algorithm uses the physics equation f=ma to apply a force to a physics body.

I had first seen this algorithm used in Unity and I have researched how ForceMode is used and I have implemented this algorithm partially in Python as I have only used the ForceModes Force, Impulse and Acceleration in my code.

The function has arguments:

* Force: This is represented using a 2D vector
* DeltaTime: The time difference between when the previous frame that was drawn and the current frame was drawn
* ForceMode: Dictates the way that the force is applied to the body



https://i.redd.it/6y8myu3ruxo71.png

Camera.UpdateScroll: This function updates the x scroll value and the y scroll value for the camera. This ensures that the player is always on the screen when playing.

*set* scroll\_x = scroll\_x + int((entity\_x - scroll\_x  - int((display\_surface\_width + target\_rect\_width) / 2)) \* 0.2)

*set* scroll\_y = scroll\_y + int((entity\_y - scroll\_y  - int((display\_surface\_height + target\_rect\_height) / 2)) \* 0.2)

*set* scroll\_x = min(max(0, scroll\_x), level\_width - display\_surface\_width)

*set* scroll\_y = min(max(0, scroll\_y), level\_height - display\_surface\_height)

Level.GenerateLevelData: This function generates a LevelData object which is a typed dictionary.

The tiles are placed into a dictionary where the key is tuple that signifies the chunk co-ordinate that the tile is in.

The position of the player and a LevelFinish object.

The LevelData object is generate from a level section in an LDTK JSON file.

The chunk co-ordinate of a tile is determined by the following algorithm:

chunk\_x = tile\_rect\_x // chunk\_size

chunk\_y = tile\_rect\_y // chunk\_size

*set* tiles = []

*set* player\_position = [-1, -1]

*foreach* (layer in layer\_instances) *do*

*switch* (layer\_identifier) *do*

*case* "AutoTiles":

*foreach* (tile in auto\_tile\_layer) *do*

*//generate\_level\_tile*

                tiles.append(level\_tile)

*end*

*break*

*case* "Tiles":

*foreach* (tile in tile\_layer) *do*

*//generate\_level\_tile*

            tiles.append(level\_tile)

*end*

*break*

*case* "Entities":

*foreach* (entity in entity\_instances) *do*

*switch* (entity\_instance\_identifier) *do*

*case* "Player":

*set* player\_position = entity\_position

*break*

*case* "Finish":

*//generate\_level\_finish\_object*

*set* level\_finish = level\_finish

*break*

*end*

*end*

*end*

*end*

**Diagram, schematic

Description automatically generated**

The player behaves differently depending on the state the player is currently in. The conditions for state transitions can be seen in the player state machine diagram.

Player State Transition Diagram

**Diagram, schematic

Description automatically generated**

Sample Inputs for Player:

xInput: “A” or “D”

dashInput: “SPACE”

jumpInput: “W”

However these controls can be changed by the user in the settings menu.

One part of the `player\_run\_state` update function calculates the force necessary to apply to the player.

This works by first calculating the speed the player is trying to reach (target\_speed) and then calculating an acceleration rate based on the target speed.

The speed difference is then calculated and the acceleration rate is multiplied by the speed difference.

This means that if the difference between the player’s current x velocity is close to the target speed then only a small force is applied.

An extra force is also applied to decrease the time it takes to change direction.

*set* target\_speed = x\_move\_input \* player\_run\_speed

*set* speed\_difference = target\_speed - player\_x\_speed

*if* abs(target\_speed) > 0 *do*

*set* acceleration\_rate = player\_acceleration\_rate

*end*

*else* *do*

*set* acceleration\_rate = player\_deceleration\_rate

*end*

*set* movement = pow(abs(speed\_difference) \* acceleration\_rate, self.player.velocity\_power) \* sign(speed\_difference)

*//add the force the player with ForceMode FORCE and pass in delta\_time*

*if* abs(target\_speed) == 0 *do*

    friction = sign(player\_x\_speed) \* min(abs(player\_x\_speed), self.player.friction

    )

*//add the friction force to the player with ForceMode IMPULSE*

*end*

Data Dictionary

|  |  |  |
| --- | --- | --- |
| Variable Name | Data Type | Use |
| config | NeaGameConfig | Stores the configuration for the game |
| debug | bool | Whether the game is being ran in debug mode |
| debug\_file | str | The name of the debug file |
| platformer\_folder | pathlib.Path | The path to root directory of the application |
| game\_folder | pathlib.Path | The path to the scripts folder |
| assets\_folder | pathlib.Path | The path to the assets\_folder |
| background\_folder | pathlib.Path | The path to where the background image layers are stored |
| gui\_folder | pathlib.Path | The path to the assets for the gui |
| music\_folder | pathlib.Path | The path to the music folder |
| player\_folder | pathlib.Path | The path to the player data folder |
| sfx\_folder | pathlib.Path | The path to sound\_effects folder |
| worlds\_folder | pathlib.Path | The path to the worlds folder |
| directories | dict[str, pathlib.Path] | A dictionary to store the paths to various folders |
| music\_volume | float | The volume of the music |
| sfx\_volume | float | The volume of the sound effects |
| nea\_game | NeaGame | Creates the root window for the application that controls which window is shown each frame |
| screen | pygame.Surface | The screen that the display\_surface is scaled onto. |
| display\_surface | pygame.Surface | A smaller surface where tiles and entities are rendered. This surface is then scaled up to the size of the screen |
| engine | Engine | Provides an interface to update and access the time difference between when the previous frame was drawn and when the current frame was drawn |
| clock | pygame.time.Clock | Allows functionality to control the framerate of the application |
| previous\_time | float | Stores the timestamp of when the previous frame was drawn |
| delta\_time | float | The time difference between when the previous frame was drawn and when the current frame was drawn |
| fps | int | Stores the fps that application is trying to run at |
| sound\_manager | SoundManager | Creates functionality for controlling the volume and sounds the background music and sound effects. |
| sound\_effects | dict[str, pygame.mixer.Sound] | Stores a dictionary of the each sound effect |
| bgm\_volume | float | Stores the volume of the background music as a float between 0 and 1 |
| sfx\_volume | float | Stores the volume of all sound effects as a float between 0 and 1 |
| current\_bgm | pathlib.Path | Stores the path to the current background music |
| sound | pygame.mixer.Sound | Stores the sound to be played on the current frame |
| active\_window | Window | Stores the active window |
| scale\_factor | int | Stores the ratio between the resolution of the screen and the resolution of the display\_surface |
| windows | dict[str, Window] | Stores each window that has been active. |
| transition | None | Callable[[], None] | Stores the current transition where the transition is a function that has no arguments and returns no value |
| transition\_start\_radius | int | Stores the starting radius of the transition |
| transition\_circle\_centre | tuple[int. int] | Stores the centre of the circle for the transition |
| current\_transition\_frame | int | Stores the frame that the current transition is on |
| is\_transitioning | bool | Stores whether the application is transitioning or not |
| is\_transition\_done | bool | Stores whether the current transition is done |
| events | list[pygame.event.Event] | Stores a list of each event that has occurred on the current frame |
| surface | pygame.Surface | The surface that the transition is drawn onto, a colourkey is then used to remove the rest of colour of the surface |
| parent | NeaGame | Stores the main application for the window |
| buttons | dict[str, Button] | Stores a dictionary of the buttons for the window |
| passive\_image | pygame.Surface | Stores the image of the button when it is not hovered over |
| active\_image | pygame.Surface | Stores the image of the button when hovered over but not clicked |
| on\_click\_image | pygame.Surface | Stores the image of the button when it is clicked |
| current\_image | pygame.Surface | Stores the current image of the button |
| rect | pygame.Rect | A Rect object for the object |
| mask | pygame.Mask | A Mask object for the object, which can be used for pixel perfect collision detection |
| can\_be\_clicked | bool | Whether the button can be clicked or not |
| clicked | bool | Whether the buttons is clicked or not on the current frame |
| click\_delay | float | The time delay between when the button is clicked and the clicked variable changes. This is used to be able to show the click animation for the button |
| click\_timer | float | The time since the button was clicked |
| mouse\_pos\_in\_mask | tuple[int. int] | The position of the mouse in the mask |
| background\_layers | list[BackgroundLayer] | Stores a list of each layer that makes up the background |
| image | pygame.Surface | Stores the image of the object |
| sine\_scale\_factor | float | The value which the sine curve for the background layer is scaled by |
| sine\_stretch\_factor | float | The value which the sine curve for the background layer is stretched by |
| sine\_translation\_factor | float | The value which the sine curve for the background layer is translated by |
| x\_scroll | float | The value which the background is scrolled by on the x-axis |
| handle\_image | pygame.Surface | A copy of the image for the layer |
| clip\_rect | pygame.Rect | The subsection of the layer which will be used for the background |
| splash\_screen\_opacity | float | The opacity of the background between 0 and 1 |
| title | Title | The title for the main menu |
| scaled\_mouse\_pos | tuple[int, int] | Stores the scaled mouse position of the mouse in the display surface |
| mouse\_clicked | bool | Stores whether the mouse has been clicked on the current frame |
| window | Window | Stores the window to be switched to |
| button\_folder\_path | pahtlib.Path | Stores the path to the button folder for the window |
| padding | tuple[int, int] | The minimum distance between the button and the edge of the screen |
| spacing | tuple[int, int] | The distance between each button on both axis |
| world\_num | int | The world number of the level to play |
| level\_num | int | The level number of the level to play |
| world\_identifier | str | The identifier of the world |
| level\_identifier | str | The identifier of the level |
| world | World | Stores the world as a world object |
| tileset | Tileset | Stores the tileset for the world |
| grid\_height | int | The height of tileset image in tiles |
| grid\_width | int | The width of the tileset image in tiles |
| grid\_size | int | The size of the tiles in pixels |
| collision\_types | dict[str, Any] | Stores the different collision types for tiles from an LDTK json file |
| tiles | dict[int, TilesetTile] | Stores the TilesetTile objects in a dictionary where the key is the identifier of the tile |
| levels | dict[str, Level] | Stores Level objects in a dictionary where the key is identifier of the level |
| data | dict[str, Any] | The JSON data from an LDTK json file |
| identifier | str | The identifier for the level |
| height | int | The height of the level in pixels |
| width | int | The width of the level in pixels |
| level\_data | LevelData | A typed dictionary to store the data about the level finish object, the position of any entities and the tiles for a level |
| render\_queue | CircularQueue[RenderObject] | Uses a circular queue structure to store each object that needs to be rendered on the current frame |
| max\_size | int | The maximum number of elements in the queue |
| queue | numpy.NDArray[Any] | The array of size max\_size that stores each element of the queue |
| front | int | The front pointer of the queue |
| rear | int | The rear pointer of the queue |
| player | Player | The player object for the game which encapsulates attributes and methods to update the player’s position and state as well as the current image of the player |
| idle\_state | PlayerIdleState | The state that the player will be in when idle |
| run\_state | PlayerRunState | The state that the player will be in when running |
| land\_state | PlayerLandState | The state that the player will be in when landing |
| dash\_state | PlayerDashState | The state that the player will be in when dashing |
| jump\_state | PlayerJumpState | The state that the player will be in when jumping |
| wall\_jump\_state | PlayerWallJumpState | The state that the player will be in when wall jumping |
| in\_air\_state | PlayerInAirState | The state that the player will be in when in the air |
| slide\_state | PlayerSlideState | The state that the player will be in when sliding |
| state\_name | str | The name of the state |
| animation\_index | int | The index of the frame for the current image of the player |
| start\_time | float | The time when the player entered the state |
| is\_exiting\_state | bool | Whether the player is exiting the state or not |
| animation\_frame\_time | float | The time each frame of the animation is displayed for |
| last\_animation\_index\_change | float | The time when the frame for the animation last changed |
| target\_speed | float | The speed that the player is trying to reach |
| speed\_difference | float | The difference between target\_speed and the player’s current x velocity |
| acceleration\_rate | float | The rate at which the player will accelerate towards the target speed |
| movement | float | The amount the player will move on the current frame |
| friction | Vector2D | The force to be applied to the player if they are trying to slow down |
| move\_input | Vector2D | The move input of the player on both axis |
| is\_ability\_done | bool | Whether the ability has finished or not |
| force | Vector2D | The force to be applied to the player |
| jump\_input\_time | float | Used for jump buffering to check if the player has pressed the jump button a certain amount of time before touching the ground |
| gravity\_scale | float | The magnitude of the force of gravity that the player will experience on the current frame |
| slide\_direction | int | The direction of the wall that the player is sliding towards |
| frames: | dict[str, list[pygame.Surface]] | A dictionary that contains each frame for an entities animation depending on their current state |
| flip\_x | bool | Whether the image should be flipped on the x axis |
| action\_space | PlayerActionSpace | Describes the number of different actions that the player can do |
| actions | dict[PlayerActionSpace, int] | The key binding for a given actions |
| actions\_performed\_on\_current\_frame | dict[PlayerActionSpace, bool] | A dictionary that holds whether a given action is pressed on the current frame |
| horizontal | int | The horizontal input of the player |
| vertical | int | The vertical input of the player |
| mass | float | The mass of the entity |
| internal\_fps | int | The internal fps of the application |
| velocity | Vector2D | The velocity vector of the entity |
| current\_state | PlayerState | The current state of the entity |
| previous\_state | PlayerState | The previous state of the entity |
| direction | int | The direction that the entity is facing on the x axis |
| old\_rect | pygame.Rect | The rect of the entity on the previous frame |
| x\_run\_speed | float | The maximum speed of the entity on the x axis |
| jump\_hang\_acceleration\_mult | float | The multiplier to the entity’s acceleration rate the entity’s y velocity is within the jump hang threshold |
| jump\_hang\_max\_speed\_mult | float | The multiplier to the entity’s target speed when the entity’s y velocity is within the jump hang threshold |
| deceleration\_rate | float | The rate at which the entity’s x velocity decrease |
| air\_acceleration\_multiplier | float | The multiplier to the entity’s acceleration rate when the entity is in the air |
| velocity | float | The power that the entity’s x velocity is raised to |
| jump\_force | float | The force of the entity’s jump |
| jump\_hang\_time\_threshold | float | The threshold to determine whether the entity is in jump hang time |
| jump\_hang\_gravity\_mult | float | The multiplier to the entity’s gravity scale when the entity’s y velocity is within the jump hang threshold |
| jump\_fast\_fall\_mult | float | The multiplier to the entity’s gravity scale when the entity’s y velocity is less than 0 |
| coyote\_time | float | The amount of time the player has to jump despite not being grounded |
| jump\_buffer\_time | float | The amount of time the where the player press the jump button before landing on the ground |
| max\_fall | float | The terminal y velocity of the entity |
| land\_animation\_time | float | The amount of time that the land animation is played for |
| wall\_jump\_force | Vector2D | The force for the entity’s wall jump |
| wall\_jump\_time | float | The amount of time the entity is in the wall jump state |
| wall\_jump\_lerp | float | The lerp value used to determine how quickly the user gains control back of the player after wall jumping |
| wall\_slide\_velcoity | float | The y velocity of the entity when sliding down a wall |
| can\_dash | bool | Determines whether the entity can dash |
| dash\_time | bool | The amount of time the entity is in the dash state |
| dash\_speed | float | The speed that the entity reaches when dashing |
| friction | float | The friction applied to the player when slowing down |
| collision | LevelTile | The LevelTile object that the player has collided into |
| display\_surface\_height | int | The height of the display\_surface in pixels |
| display\_surface\_width | int | The width of the display\_surface in pixels |
| scroll\_x | int | The amount that the camera is scrolled by on the x axis |
| scroll\_y | int | The amount that the camera is scrolled by on the y axis |
| new\_settings\_json | dict[str, Any] | The new data for the config.json file |
| visible\_chunks | list[list[LevelTile]] | The list of the tiles in the visible chunks |
| render\_object | RenderObject | The object to be rendered onto the display\_surface |
| title\_image | pygame.Surface | The image of the title for the window |
| text | pygame.Surface | The text for the window |
| music\_icon | pygame.Surface | The music icon |
| sound\_icon | pygame.Surface | The sound icon |
| action\_buttons | dict[str, ActionButton] | Stores a dictionary of the action buttons for the window |
| sliders | dict[str, Slider] | Stores a dictionary of the sliders for the window |
| key\_bindings | list[int] | The keybindings that correspond to the various actions that the player can take |
|  |  |  |

File Structure

NeaGame

│ .gitattributes

│ .gitignore

│ app.py

│ main.py

│ README.md

│

├───assets

│ │ ChunkFive-Regular.otf

│ │ music.ceol

│ │ PALETTE.gpl

│ │ title\_font.ttf

│ │

│ ├───background

│ │ └───sky\_mountain

│ │ -1.png

│ │ 0.png

│ │ 1.png

│ │ 2.png

│ │ 3.png

│ │ 4.png

│ │ 5.png

│ │

│ ├───gui

│ │ ├───level\_selection

│ │ │ ├───buttons

│ │ │ │ ├───1-1

│ │ │ │ │ 0.png

│ │ │ │ │ 1.png

│ │ │ │ │ 2.png

│ │ │ │ │

│ │ │ │ ├───1-2

│ │ │ │ │ 0.png

│ │ │ │ │ 1.png

│ │ │ │ │ 2.png

│ │ │ │ │

│ │ │ │ ├───1-3

│ │ │ │ │ 0.png

│ │ │ │ │ 1.png

│ │ │ │ │ 2.png

│ │ │ │ │

│ │ │ │ └───back

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │ 2.png

│ │ │ │

│ │ │ └───settings

│ │ │ └───back

│ │ │ 0.png

│ │ │ 1.png

│ │ │ 2.png

│ │ │

│ │ ├───main\_menu

│ │ │ │ title.png

│ │ │ │

│ │ │ └───buttons

│ │ │ ├───exit

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │ 2.png

│ │ │ │

│ │ │ ├───play\_game

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │ 2.png

│ │ │ │

│ │ │ └───settings

│ │ │ 0.png

│ │ │ 1.png

│ │ │ 2.png

│ │ │

│ │ ├───pause

│ │ │ └───buttons

│ │ │ ├───exit

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │ 2.png

│ │ │ │

│ │ │ └───resume

│ │ │ 0.png

│ │ │ 1.png

│ │ │ 2.png

│ │ │

│ │ └───settings

│ │ │ music.png

│ │ │ sound.png

│ │ │ text.png

│ │ │ title.png

│ │ │

│ │ ├───action\_buttons

│ │ │ ├───0

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │

│ │ │ ├───1

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │

│ │ │ ├───2

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │

│ │ │ ├───3

│ │ │ │ 0.png

│ │ │ │ 1.png

│ │ │ │

│ │ │ └───4

│ │ │ 0.png

│ │ │ 1.png

│ │ │

│ │ ├───buttons

│ │ │ └───back

│ │ │ 0.png

│ │ │ 1.png

│ │ │ 2.png

│ │ │

│ │ ├───keys

│ │ │ 1-key.png

│ │ │ 2-key.png

│ │ │ 3-key.png

│ │ │ 4-key.png

│ │ │ 5-key.png

│ │ │ 6-key.png

│ │ │ 7-key.png

│ │ │ 8-key.png

│ │ │ 9-key.png

│ │ │ a-key.png

│ │ │ b-key.png

│ │ │ blank-lg-key.png

│ │ │ blank-md-key.png

│ │ │ blank-sm-key.png

│ │ │ c-key.png

│ │ │ d-key.png

│ │ │ e-key.png

│ │ │ f-key.png

│ │ │ g-key.png

│ │ │ h-key.png

│ │ │ i-key.png

│ │ │ j-key.png

│ │ │ k-key.png

│ │ │ l-key.png

│ │ │ left\_alt-key.png

│ │ │ left\_ctrl-key.png

│ │ │ left\_shift-key.png

│ │ │ m-key.png

│ │ │ n-key.png

│ │ │ naaB.png

│ │ │ o-key.png

│ │ │ p-key.png

│ │ │ q-key.png

│ │ │ r-key.png

│ │ │ s-key.png

│ │ │ space-key.png

│ │ │ t-key.png

│ │ │ thumbs.db

│ │ │ u-key.png

│ │ │ v-key.png

│ │ │ w-key.png

│ │ │ x-key.png

│ │ │ y-key.png

│ │ │ z-key.png

│ │ │

│ │ └───sliders

│ │ └───volume

│ │ 0.png

│ │ 1.png

│ │ 2.png

│ │

│ ├───music

│ │ bgm.wav

│ │

│ ├───player

│ │ │ sprite.png

│ │ │

│ │ ├───Dash

│ │ │ 0.png

│ │ │

│ │ ├───Idle

│ │ │ 0.png

│ │ │

│ │ ├───InAir

│ │ │ 0.png

│ │ │

│ │ ├───Jump

│ │ │ 0.png

│ │ │

│ │ ├───Land

│ │ │ 0.png

│ │ │

│ │ ├───Run

│ │ │ 0.png

│ │ │ 1.png

│ │ │ 2.png

│ │ │

│ │ ├───Slide

│ │ │ 0.png

│ │ │

│ │ └───WallJump

│ │ 0.png

│ │

│ ├───sfx

│ │ click.wav

│ │ jump.wav

│ │

│ ├───tiles

│ │ brick\_0.png

│ │ brick\_1.png

│ │ dirt\_0.png

│ │ dirt\_1.png

│ │ dirt\_2.png

│ │ dirt\_3.png

│ │ dirt\_4.png

│ │ dirt\_5.png

│ │ grass\_0.png

│ │ grass\_1.png

│ │ grass\_2.png

│ │ platform\_0.png

│ │ platform\_1.png

│ │ platform\_2.png

│ │ spike\_0.png

│ │ spike\_1.png

│ │ spike\_2.png

│ │ spike\_3.png

│ │ tileset.png

│ │

│ └───worlds

│ 1.json

│

├───diagrams

│ aggregation\_diagram.drawio

│ class\_diagram.drawio

│ inheritance\_diagram.drawio

│

├───NeaGameDocs

│ Analysis.docx

│ Code.docx

│ Design.docx

│

└───nea\_game

│ circular\_queue.py

│ config.json

│ config.py

│ design\_algos.pseudo

│ nea\_game.py

│ sound\_manager.py

│ \_\_init\_\_.py

│

├───calc

│ lerp.py

│ near\_zero.py

│ sign.py

│ vector2d.py

│ \_\_init\_\_.py

│

├───components

│ input.py

│ renderer.py

│ rigidbody2d.py

│ \_\_init\_\_.py

│

├───entity

│ base\_entity.py

│ level\_finish.py

│ \_\_init\_\_.py

│

├───game

│ camera.py

│ engine.py

│ game.py

│ render\_object.py

│ \_\_init\_\_.py

│

├───gui

│ button.py

│ root.py

│ slider.py

│ title.py

│ window.py

│ \_\_init\_\_.py

│

├───ldtk\_world\_loader

│ collision\_type.py

│ level.py

│ level\_data.py

│ level\_tile.py

│ tileset.py

│ tileset\_tile.py

│ world.py

│ \_\_init\_\_.py

│

├───menu

│ action\_button.py

│ background\_layer.py

│ level\_selection.py

│ main\_menu.py

│ pause\_menu.py

│ settings\_menu.py

│ \_\_init\_\_.py

│

├───player

│ │ player.py

│ │ player\_action\_space.py

│ │ \_\_init\_\_.py

│ │

│ ├───sub\_states

│ │ player\_dash\_state.py

│ │ player\_idle\_state.py

│ │ player\_in\_air\_state.py

│ │ player\_jump\_state.py

│ │ player\_land\_state.py

│ │ player\_run\_state.py

│ │ player\_slide\_state.py

│ │ player\_wall\_jump\_state.py

│ │ \_\_init\_\_.py

│ │

│ └───super\_states

│ player\_ability\_state.py

│ player\_grounded\_state.py

│ \_\_init\_\_.py

│

└───states

player\_state.py

player\_state\_machine.py

\_\_init\_\_.py

GUI Overview

Main Menu

A screenshot of a computer

Description automatically generated with low confidenceThere is a scrolling mountain background with buttons that have different images when the mouse is hovered over them and there is a click animation. There is also music playing in the background.

A screenshot of a computer

Description automatically generated with low confidence

A picture containing shoji, building, crossword puzzle

Description automatically generated

The Colour Palette used to Create a Blue and Grey Mountain Range for the Main Menu with Buttons

Level Selection

The screen where the user can select the level want to play. The buttons also have a different image when hovered over and only the levels that the user has unlocked change colour when hovered over and can be clicked. There is also a back button which the user can use to return to the main menu.

A picture containing graphical user interface

Description automatically generated

A picture containing icon

Description automatically generatedSettings

The Colour Palette used to Create the Level Selection Screen

The user is able to change the controls for the actions that the player can perform as well as the volume for both sound effects and music.

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

A picture containing shoji, crossword puzzle

Description automatically generated

The Colour Palette used for the Settings Menu

Libraries

The libraries required to build the project are Pygame, Numpy. The rest of libraries used are part of the libraries are part of the Python 3.11 standard library so do not need to be installed if an up to date version of Python is present.

Pygame is used to render images onto the screen as well as to interact with the hardware of the computer involved in getting both keyboard and mouse input from the user and using the speakers to play sounds.

Numpy is used to create an array of fixed size which is used to create the circular queue class. This means that queue operations can be implemented traditional as well as with the high access speeds.