**Python**



(Concentrate)

Major Project Documentation

Abram Liddell – *12IST*

Mr Anthony Jarvis

Table of Contents

[Defining and Understanding the Problem 3](#_Toc65817454)

[Statement and Scenario 3](#_Toc65817455)

[Needs for the Project 4](#_Toc65817456)

[Functionality Requirements 4](#_Toc65817457)

[Compatibility Issues 4](#_Toc65817458)

[Performance Issues 4](#_Toc65817459)

[Issues Relevant to the Proposed Solution 4](#_Toc65817460)

[Planning and Designing Software Solutions 5](#_Toc65817461)

[System Flowchart 5](#_Toc65817462)

[Context Diagram 6](#_Toc65817463)

[Data Flow Diagram 7](#_Toc65817464)

[Input Process Output (IPO) Diagrams 8](#_Toc65817465)

[Data Dictionary 12](#_Toc65817466)

[Screen Design 15](#_Toc65817467)

[Storyboard 18](#_Toc65817468)

[Planning and Designing (Additional Resources) 19](#_Toc65817469)

[Gantt Chart 19](#_Toc65817470)

[Logbook 19](#_Toc65817471)

[Concept Prototype 20](#_Toc65817472)

# Defining and Understanding the Problem

## Statement and Scenario

Many 2D games have been created that emulate a run and gun game. In recent years, similar games have been made that implement the use of borders as a “kill zone”.

Concentrate is a game that combines these two characteristics in a way that enforces the user to be both accurate and to endure as long as possible. The user can control a character with the WASD keys and has the option of a few game modes. The first game mode (to be featured in the prototype) is an endless game mode in which the user attempts to avoid the borders and control their sprite for as long as possible. The second game mode (not featured in the prototype) allows the user to change the speed of the game by relating their x-coordinate with the speed of the game. As the user approaches the right edge of the screen, the map autogenerates faster. One can achieve a high score by getting as far to the edge as possible, and therefore reducing the time it takes to complete a pre-set track. The final game mode (also not featured in the prototype) is an infinite game mode in which the speed increases automatically and linearly and the player tries to last as long as possible.

This major project is to be based on the Python programming language and will require the use of a library called PyGame for most of the important game functions.

Based on previous experience, this solution seems feasible; unfortunately, one trade-off in the ease of use of PyGame is that it does not support VSync. This can cause screen tearing at a certain repeating interval (the length of this interval depends on a variety of factors). Since Python can support high-intensity CPU and GPU operations, it should have no problem operating a program that will use a low to moderate amount of trigonometry and track up to 10 objects at once. The use of pixel art (Piskel) is to be used repeatedly throughout the project as a creation tool for sprites and backgrounds, and other image generation websites/software used will be referenced in the final version of this documentation.

## Needs for the Project

* Main Menu.
* Config Page.
* Instructions.
* An autogenerating map.
* Border collision implementation.
* A customisable, movable character.
* Parallax scrolling for the background.
* A timer for timed games.
* A way of generating and storing a game to be played in accordance with the second game mode listed in the Statement and Scenario.
* A way of changing the speed and frame rate of the game.
* Audio that well represents the game status.

## Functionality Requirements

* Run at or above 60FPS.
* Run endlessly where required.
* Prevent excess RAM usage.
* Minimal amount of screen tearing and “glitchiness”.

## Compatibility Issues

* PyGame graphics may display differently on Linux or MacOS compared to Windows 10.
* Users with different Python versions installed may have issues compiling the software.
* Users with different PyGame versions installed may have issues compiling the software.
* A .exe file will not work on a platform other than Windows 10.

## Performance Issues

* The game may lag or glitch on older generations of computers, particularly on machines with limited resources.
* VSync cannot be implemented, so the screen will tear periodically.
* The game will natively run at 220FPS. This frame rate is chosen higher than most the frequency of most displays to minimise tearing and maximise smoothness.

## Issues Relevant to the Proposed Solution

* Using the WASD keys may be harder than using the mouse for some people. An option to enable mouse usage could be implemented as an option in the final version.
* A background made entirely in Piskel may be difficult considering the size and the gif-like repeatability that is required as it parallax scrolls.
* Python file handling is time consuming and resource intensive. As such, recording a game may not be possible. This means that the game needs to be autogenerated BEFORE the player starts the game if they wish to play it again or improve on a time high score.
* Using trigonometry for collision detection will be CPU-intensive; particularly at 220FPS. The solution is to have borders that draw themselves and then detect a collision, but that requires the handling of at least double the number of sprites already on the screen. It would also be more difficult to implement.
* To compensate for the CPU intensiveness of the trigonometry to detect collisions, the rest of the code will need to run efficiently as possible. This will require regular refactoring throughout the development process.

# Planning and Designing Software Solutions

## System Flowchart

Diagram, schematic

Description automatically generated

## Context Diagram

Diagram

Description automatically generated

## Data Flow Diagram

Diagram

Description automatically generated

## Input Process Output (IPO) Diagrams

**NOTE: *These all appear chronologically as they are called throughout the program.***

|  |  |  |
| --- | --- | --- |
| **startupChecks()** | | |
| Input | Process | Output |
|  | Check that essential pygame modules are loaded. If not, display warnings to the user. | Print “WARNING: Fonts Disabled.”  And/or  Print “WARNING: Sounds Disabled.” |

|  |  |  |
| --- | --- | --- |
| **main()** | | |
| Input | Process | Output |
|  | Initialize borderRequired == True |  |
|  | Initialize screen |  |
|  | Create window and set caption to “Concentrate” | Window with caption “Concentrate” |
|  | Create mainMenuSurface |  |
|  | Create configMenuSurface |  |
|  | Create instructionMenuSurface |  |
|  | Create gameSurface |  |
|  | While gameStarted == False: mainMenu() |  |
|  | configMenu() |  |
|  |  | Print “GAME STATUS: Running.” |
|  | drawGame() |  |
|  | Initialize sounds = Sounds() |  |
|  | Initialize player = Player() |  |
|  | Initialize border = Border() |  |
|  | Initialize allSprites player renderer |  |
|  | Create borderGroup group |  |
|  | Add border to borderGroup |  |
|  | Add player to allSprites |  |
|  | Blit gameSurface to the screen |  |
|  | Draw allSprites group to screen |  |
|  | Display screen in window (pygame.display.flip) |  |
|  | Initialize playerMoveX = 0 |  |
|  | Initialize playerMoveY = 0 |  |
|  | Initialize speed = 3 |  |
|  | Initialize fps\_count = 0 |  |
| Player input pygame.quit() | Exit the game | Window closes |
| Player input (WASD keys) | playerMoveX +/- speed (depending on keys pressed) |  |
|  | playerMoveY +/- speed (depending on keys pressed) |  |
|  | player.update((playerMoveX, playerMoveY)) |  |
|  | borderGroup.update() |  |
|  | If borderRequired == True:  Create new border called border, add it to borderGroup.  Also loop through all current borders. If one of their X-coordinates < 100, then delete that border. | Print “STATUS: Border Added.” |
|  | Blit gameSurface to the screen |  |
|  | Draw borderGroup to the screen |  |
|  | Draw allSprites to the screen |  |
|  | Display screen in window | Display the game |
|  | Pygame.clock.tick(220) (regulates the while loop to 220 times per second) |  |

|  |  |  |
| --- | --- | --- |
| **mainMenu()** | | |
| Input | Process | Output |
|  | Make mainMenuSurface white |  |
|  | Add title text and blit to mainMenuSurface |  |
|  | Add Start Game text and blit to mainMenuSurface |  |
|  | Add Instructions text and blit to mainMenuSurface |  |
|  | Display mainMenuSurface to the screen |  |
|  | Display the screen to the window | Displays the main menu |
|  | Initialize menuGoing = True |  |
| mousePos | While menuGoing == True, get mouse position. If mouse click happens, and mousePos is within the bounds of “Start Game” text,  menuGoing = False and gameStarted = True. | Print “CONTROL: Start Game” |
|  | If mousePos is within the bounds of “Instructions” text, call instructionMenu() and set menuGoing = False | Print “CONTROL: Instructions” |

|  |  |  |
| --- | --- | --- |
| **instructionMenu()** | | |
| Input | Process | Output |
|  | Make instructionMenuSurface white |  |
|  | Add title text and blit to instructionMenuSurface |  |
|  | Add back text and blit to instructionMenuSurface |  |
|  | Display instructionMenuSurface to the screen |  |
|  | Display the screen to the window | Displays the instruction menu to the screen |
|  | Initialize menuGoing = True |  |
|  | While menuGoing == True, get mouse position. If mouse click happens, and mousePos is within the bounds of “Back” text,  menuGoing = False. | Print “CONTROL: Back” |

|  |  |  |
| --- | --- | --- |
| **configMenu()** | | |
| Input | Process | Output |
|  | Make configMenuSurface white |  |
|  | Add title text and blit to configMenuSurface |  |
|  | Add background and blit to configMenuSurface |  |
|  | Add “Start Game” text and blit to configMenuSurface |  |
|  | Display configMenuSurface to the screen |  |
|  | Display the screen to the window | Displays the configuration menu |
|  | Initialize menuGoing = True |  |
|  | While menuGoing == True, get mouse position. If mouse click happens, and mousePos is within the bounds of “Start Game” text,  menuGoing = False. | Print “CONTROL: Config Start” |

|  |  |  |
| --- | --- | --- |
| **drawGame()** | | |
| Input | Process | Output |
|  | Make gameSurface white and blit to the screen. Draw screen to window. | Window with white background. |

|  |  |  |
| --- | --- | --- |
| **Sounds()** | | |
| Input | Process | Output |
|  | Initialize an object Sounds with default values |  |

|  |  |  |
| --- | --- | --- |
| **load\_sound()** | | |
| Input | Process | Output |
| name |  |  |
| Sound files | Load sound file from filesystem into RAM |  |

|  |  |  |
| --- | --- | --- |
| **Player()** | | |
| Input | Process | Output |
|  | Initialize an object Player with default values |  |

|  |  |  |
| --- | --- | --- |
| **load\_sprite()** | | |
| Input | Process | Output |
| name |  |  |
| Image files | Load images into RAM as sprites |  |

|  |  |  |
| --- | --- | --- |
| **Border()** | | |
| Input | Process | Output |
|  | Initialize an object Border with default values |  |
| **Player.update()** | | |
| Input | Process | Output |
| playerMoveX |  |  |
| playerMoveY | Update player object position |  |

|  |  |  |
| --- | --- | --- |
| **Border.update()** | | |
| Input | Process | Output |
|  | Move border to the left at a specified speed. |  |

|  |  |  |
| --- | --- | --- |
| **generateGame()** | | |
| Input | Process | Output |
| tickNo | Run border.update() for a specified period of ticks and store it. | gameArray[] |

|  |  |  |
| --- | --- | --- |
| **saveGame()** | | |
| Input | Process | Output |
| gameArray[] | Convert gameArray[] to a save format and save to text file |  |

|  |  |  |
| --- | --- | --- |
| **loadGame()** | | |
| Input | Process | Output |
| filePath | Take text file from filePath and convert to gameArray | gameArray[] |

|  |  |  |
| --- | --- | --- |
| **runGeneratedGame()** | | |
| Input | Process | Output |
| gameArray[] | Use border.update(), but replace all values with preset values from gameArray[]. |  |

*Border.update() is featured here instead of borderGroup.update(). Since there are multiple borders, and since the borderGroup is essentially going through each border object and completing “Border.update()” for every border, this particular IPO diagram is still correct.*

## Data Dictionary

|  |  |  |
| --- | --- | --- |
| Variable Name | Purpose | Type |
| main\_dir | The main directory | string |
| assets\_dir | The assets directory | string |
| sounds\_dir | The sounds directory | string |
| images\_dir | The images directory | string |
| sprites\_dir | The sprites directory | string |
| gameStarted | A flag for use in the menu loops. | Boolean |
| borderSpeed | The speed of which the borders advance every tick. | int |
| oldY | Used in calculating the next border pathway placement. | int |
| fullname | Used in load\_sprites() and load\_images() as the name for an image and its path. | string |
| name | Used in load\_sprites(), load\_sound() and load\_images() as the name for an image. | string |
| image | The image used in load\_sprites(), load\_images(), Player() and Border() functions and classes as a storage medium for images. | Pygame image object |
| colorkey | Used in load\_sprites() and load\_images() as the determining variable for the colorkey with which to load an image. | int |
| sound | Used in load\_sound() as a temporary storage medium. | Pygame sound object |
| rect | Used in Border() and Player() as an indication of the outline of the object. | Pygame array |
| area | Used in Border() and Player() as the area of the screen. | int |
| move[] | Used to determine movement of Player() object. | array |
| width | Used to store the width of each Border() passageway. | int |
| borderRequired | A flag used to determine whether a new border needs to spawn. | boolean |
| deathSound | Sound played upon death. | Pygame sound object |
| timerSound | Sound played periodically to indicate timer. | Pygame sound object |
| startSound | Sound played at the start of the game. | Pygame sound object |
| buttonSound | Sound played when buttons are clicked. | Pygame sound object |
| gameMusicSound | Music played during the game | Pygame sound object |
| menuMusicSound | Music played in the menu | Pygame sound object |
| x | X location in the blit\_alpha() function. | Int |
| y | Y location in the blit\_alpha() function. | int |
| location[] | Array for the x and y values in the blit\_alpha() function. | array |
| temp | Temporary variable to store image. - blit\_alpha(). | Pygame image object |
| target | Surface which is being blitted to - blit\_alpha(). | string |
| opacity | Used to determine image opacity - blit\_alpha(). | int |
| screen | The screen which is blitted to the window | Pygame surface object |
| clock | Clock used to record time scores. | Pygame clock object |
| mainMenuSurface | Surface of the main menu to be blitted to surface. | Pygame surface object |
| instructionMenuSurface | Surface of the instruction menu to be blitted to surface. | Pygame surface object |
| configMenuSurface | Surface of the config menu to be blitted to surface. | Pygame surface object |
| gameSurface | Surface of the game to be blitted to the surface. | Pygame surface object |
| font | Variable with desired font. mainMenu(), instructionMenu(), configMenu(), endgameMenu() | string |
| text | Variable with desired text.  mainMenu(), instructionMenu(), configMenu(), endgameMenu() | string |
| textpos[] | Position of text.  mainMenu(), instructionMenu(), configMenu(), endgameMenu() | array |
| startpos[] | Position of start game text.  mainMenu(), configMenu(), | array |
| instructionpos[] | Position of instruction text.  mainMenu() | array |
| menuGoing | A flag used to break out of menu loops.  mainMenu(), instructionMenu(), configMenu(), endgameMenu() | boolean |
| mousepos[] | The position of the mouse  mainMenu(), instructionMenu(), configMenu(), endgameMenu() | array |
| configBackgroundImage | Variable storing background image for configuration menu | Pygame image object |
| mainMenuBackgroundImage | Variable storing background image for main menu | Pygame image object |
| instructionBackgroundImage | Variable storing background image for instruction menu | Pygame image object |
| gameArray[] | Array holding all the details for border generation of a game | array |
| gameDetails[] | Array holding all the characteristics of a game | array |

## Screen DesignDiagram Description automatically generated

Text

Description automatically generated with medium confidence

## Storyboard

A picture containing text, whiteboard

Description automatically generated

# Planning and Designing (Additional Resources)

## Gantt Chart

A screenshot of a video game

Description automatically generated

## Logbook

This excel worksheet could not be imported into this document due to its size. As such, an image is pasted below, or a hyperlink to the spreadsheet is also available:

<Logbook.xlsx>

Text

Description automatically generated

GitHub also has some of the commits from the code. These may prove to have some more detail regarding code development. It also has all documentation and resources backed up in case of file corruption or loss of data. This is the link to the repository ->

## Concept Prototype

To run the game, run the .exe file placed in the “***/Major Project/Executable/dist/***” folder. If that does not work, use the main.py file to run it. In this case, you will require Python 3.8.5, PyGame 1.9.6 and VSCode to be preinstalled.