Game Project

AS91906 and AS91907 - Do a big programming project and plan it well.

# Concept

I want to make an arcade style game where the player must navigate through many randomly generated obstacles as they move down the screen towards the player. The frequency of these obstacles will increase as the player continues upwards. The player can fire a thruster to move faster but cannot stop moving.

The player will be able to shoot up at obstacles and move horizontally but both will deplete the player’s ‘energy’ which regenerates at a set rate. If the player’s energy falls below a set amount their manoeuvrability will be limited, and they will not be able to shoot.

The aim of the game is to get as far through the obstacles as possible.

I will develop this game using Python3.8 and PyGame as I have experience with these, and I know that they have all the tools necessary to build this game.

## Suitability for audience

I want to make this game suitable for a wide range of ages and experience levels, so I need to ensure that the game is appropriate both for children and adults. I will also aim to use symbols in menus to make the game accessible to those that do not speak English well.

To allow players with various levels of experience to enjoy the game I want to implement difficulty options that the player can select from ranging from easy to extremely hard. I want the easiest version to be playable for children with minimal experience with games, and I want the hardest levels to challenge very experienced players.

The gameplay and control need to be intuitive to those with gaming experience and to those without. So I will need to make use of well-established conventions for gameplay and control so that I don’t confuse players that have lots of experience in other games and will expect mine to work in a certain way. I will also need to make sure that these conventions can be understood by new players without outside instructions.

# Components

Because of the program’s complexity it would be very hard to debug issues if I write the entire program before doing significant testing. To make sure that issues are manageable I need to test modules individually as I complete them. This also means that I need to plan the order in which to develop modules.

I will develop the functions for displaying the menu and game, as well as the code for switching between them. These parts will establish the overall structure of the program, so writing and testing them first will ensure that I put all the other parts of the program in the right places.

## Player

The player will be a class that inherits methods from the pygame sprite class. It will primarily handle moving and drawing the player on screen and will need the following methods:

Initialisation method to set up all the variables used by the player including placing it on the screen and reading and sprites from files.

An update position method that looks at the state of the input keys and energy bar to update the player’s position.

Drawing the player will be handled by pygame’s sprite functions and will be done each cycle in the main loop.

A method to check if the player is colliding with any obstacles. This will make use of the sprite functions to quickly check collisions against all the obstacles which will be other sprites stored in a separate container.

A destroy method called when the player collides with an obstacle. This will handle animating the players destruction and removing them from the screen.

## Obstacles

These will also be a class with inheritance from the pygame sprite class. Because there will be many obstacle objects in the game at once I need to make the obstacles as simple as possible to keep the game running smoothly.

## Menu loop

I will need a separate loop for the menu that listens to the players keyboard inputs and selects menu items accordingly. The menu loop should have two states, a main state that has three options: play, controls, and exit. Its second state should be a short list of the games key bindings.

## Game loop

This will handle playing the game. It will take user input and take actions based on it. It will call all the other functions needed for the game including updating game objects and drawing them on screen.

# Version control

As I am developing this project, I need to keep regular backups of the files and I need a way to revert to previous versions when I make mistakes.

For this I have decided to use GitHub. GitHub backs my project up to its servers and allows me to apply changes to the backups as versions. These versions can be easily reverted so If I try something that doesn’t work I can revert to a previous version and try again.

# Testing plan

The game’s main mechanics need to be tested to ensure that they are enjoyable for players.

## Player control

One of the key mechanics is the ship’s control, the player must always keep moving but can briefly accelerate. I have previously made a game with a very similar control mechanic but my testing of this was mostly limited to myself playing it as this was a small hobby project and was not made public. To make sure this control scheme works I will do more testing with a variety of people. I may also test a version of the game with different movement speeds to see if people find it more enjoyable. I will make sure that I get testers from multiple different demographics and experience levels

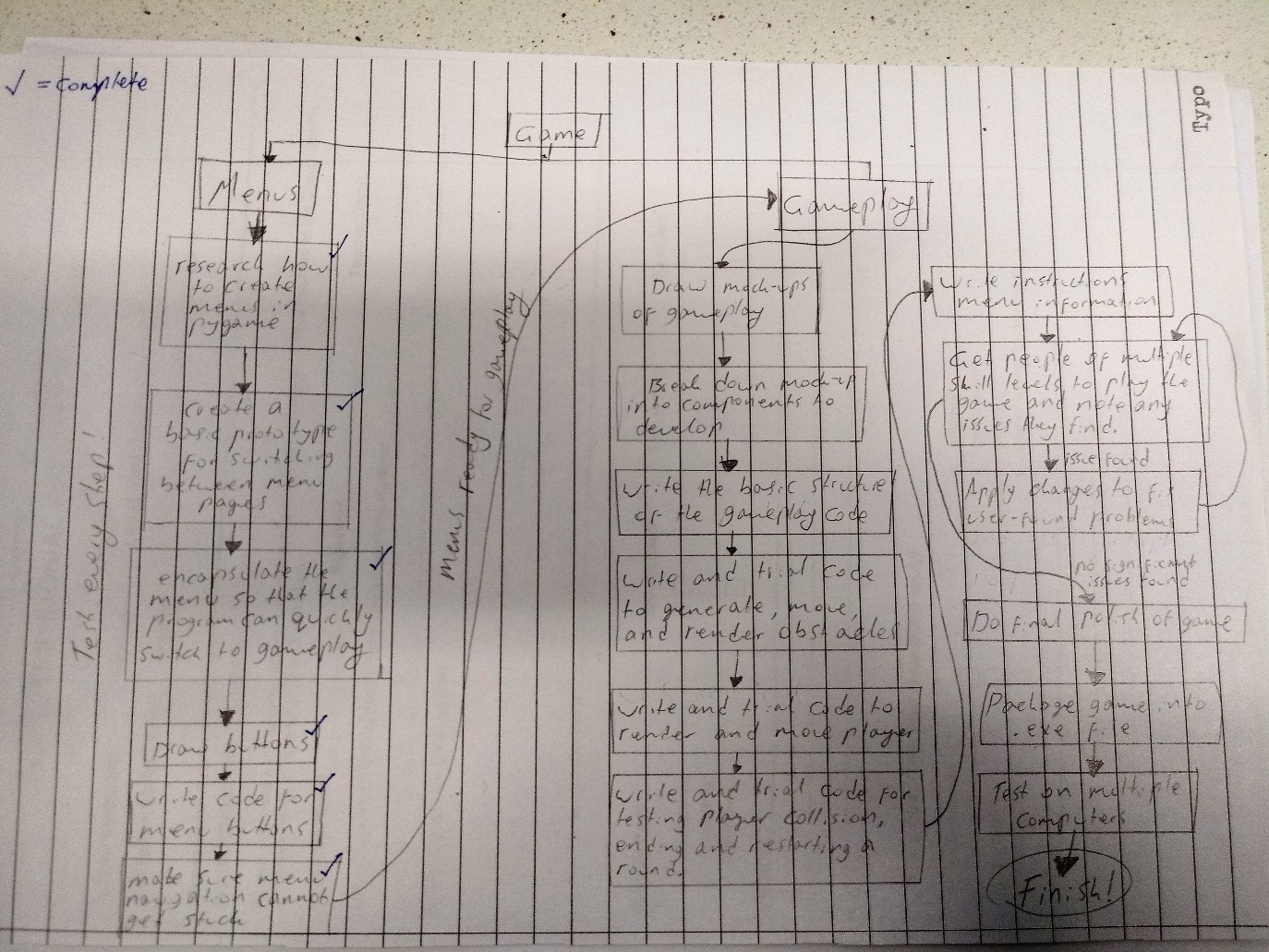
## Difficulty levels

I will need to get players from many ages and experience levels to try the game at different difficulty levels to ensure that I have enough variance in the difficulty to make the game fun for the majority of people.

## Intuitive gameplay

To make sure that the game is intuitive to a wide range of players I will ask people to play the game without giving them instructions. If they can play the game with minimal initial confusion, then the game has satisfied the requirement of being intuitive. If the players need instruction to play or enjoy the game, then I will record where they had difficulty and change the problematic elements until players find the game easy to learn.

# Overall Project plan

Due to the size of the project I need a plan for what tasks I should get done and in what order. So I drew this to show the flow of the project. This gives me a much better idea of how much I need to get done and is handy for quickly being able to see what I should do right now.

Apologies for the shoddy picture

# Menu planning

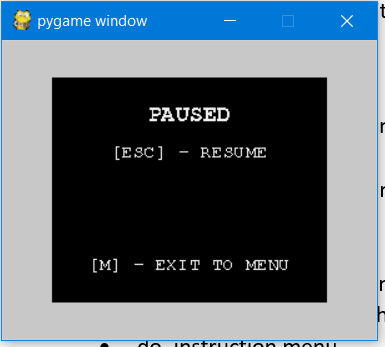
Pygame does not have a built-in way of handling menus so I need to make my own. I was unsure about how to manage all the different menu screens, so I did some research into pygame menu handling. I found this video that gave me the idea to have each menu screen as its own function that would handle rendering its unique elements and handling relevant user input from the keyboard and mouse. It also showed me that I could detect button presses by checking if the mouse pointer is over an element’s rect and then checking if the mouse button is pressed.

I decided to make the menu a class with methods for drawing each menu screen as well as supporting methods for rendering and for selecting which screen to display.

## First prototype

I will first make a very minimal menu as a prototype to work out how to switch between screens. This will also reveal the flaws in my initial plan so that I can make a better final outcome.

### Methods

* \_\_init\_\_  
  defines all the variables and gui elements needed for the menus
* do  
  looks at the menu state and then calls the method to draw that screen.  
  called externally
* do\_main\_menu  
  draws all the text for the main window and then loops through the event list and responds to relevant ones. Switches the state of the menus or game accordingly.
* do\_instruction menu  
  same as above but with different text
* do\_pause\_menu  
  “”

### Testing

|  |  |  |
| --- | --- | --- |
| **Test** | **Expected result** | **Pass** |
| Try switching between all menus | Menus are switched quickly. Every sub-menu can be exited |  |
| Go into game, pause, resume, exit to menu | Game switches states quickly and without artifacts. Switching between all menus works |  |
| Press escape on main menu | Exits program |  |

## Buttons

The first prototype used keyboard shortcuts to move between screens. This is easy to implement but not intuitive for users, so I needed to make a better interface. I want to do this by replacing keyboard shortcuts with buttons.

These buttons need to provide visual feedback to the user. I want them to have three states: idle, hover, and pressed. The hover state will make the button appear slightly different when the mouse hovers over it, this will confirm to the user that this is a button and makes the menu more interesting. The pressed state shows the button depressed for as long as the mouse button is held down for. I want the buttons to execute their linked command when the mouse button is released.

I learnt from the first prototype that pygame elements such as text take a lot of code to create so I need a good way to organize this. I will make the buttons a class that has a method for updating the button’s state, and a method for rendering the button.

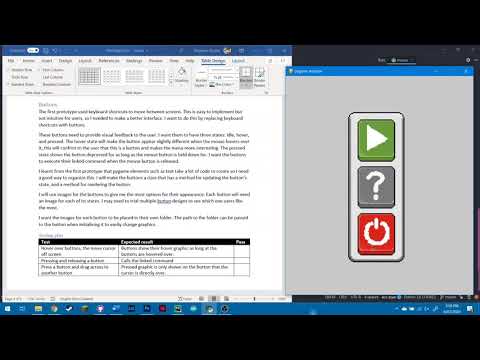
I will use images for the buttons to give me the most options for their appearance. Each button will need an image for each of its states. I may need to trial multiple button designs to see which one users like the most.

I want the images for each button to be placed in their own folder. The path to the folder can be passed to the button when initialising it to easily change graphics.

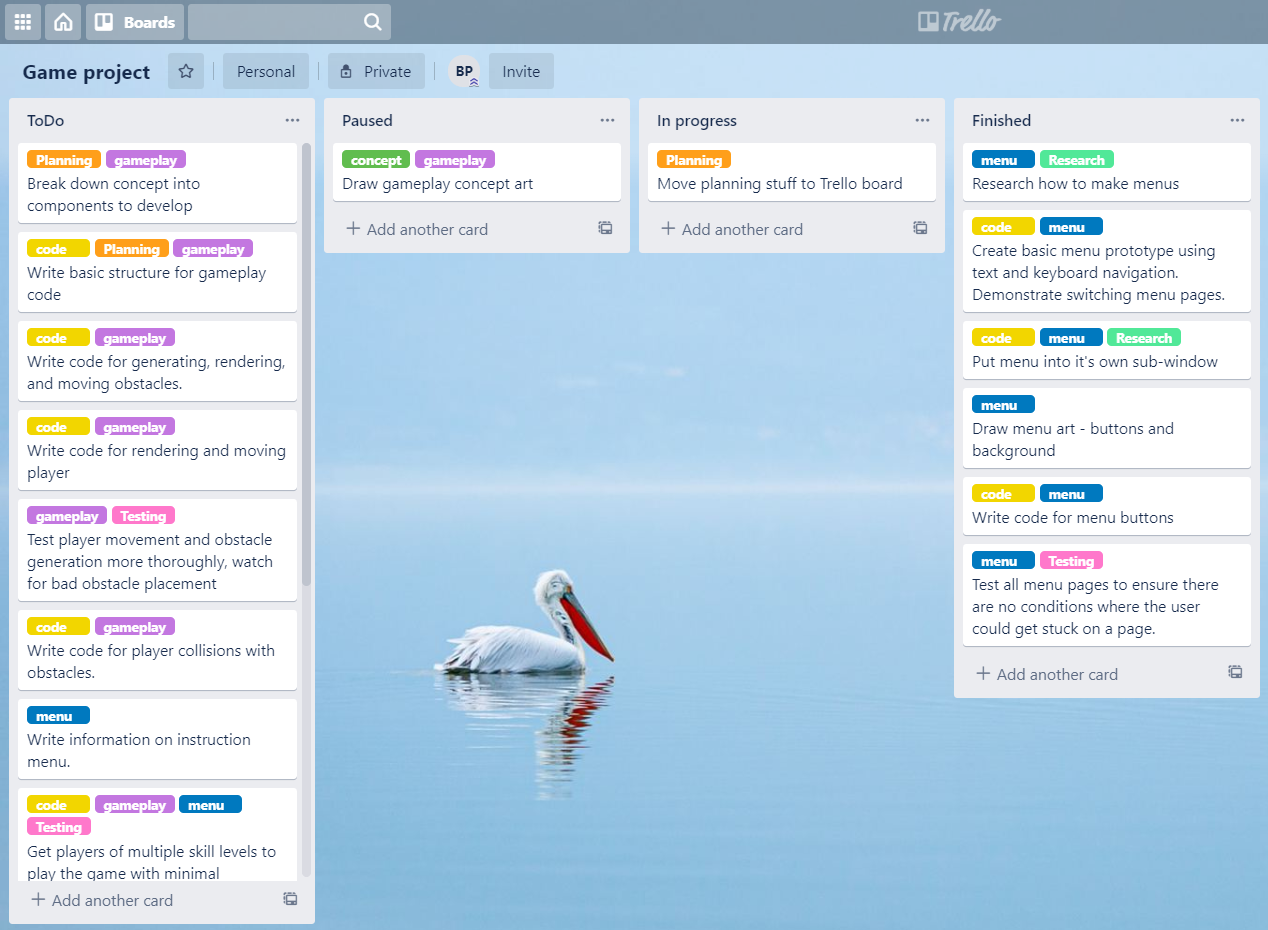
### Testing plan

|  |  |  |
| --- | --- | --- |
| **Test** | **Expected result** | **Pass** |
| Hover over buttons, the move cursor off screen | Buttons show their hover graphic as long at the buttons are hovered over. |  |
| Pressing and releasing a button | Calls the linked command |  |
| Press a button and drag across to another button. | Pressed graphic is only shown on the button that the cursor is directly over. |  |

### Menu testing video

[](https://www.youtube.com/embed/-tuKJza5L_A?feature=oembed)

# Trello Board

The paper plan I had drawn for the project was useful for initial planning but was inconvenient for working on the project. I decide

d to shift the planning to a Trello Board that would allow me to manage tasks more easily.

I added labels to the ‘cards’ so I could easily see what part of the project each card was in and what kind of work it would require.

In previous projects I have found that the In Progress list gets cluttered with partially complete, paused tasks. To avoid this issue I added a Paused list to put any tasks that I am halfway through and will come back to later.

# Obstacle planning

Each obstacle will be randomly assigned a big, medium, or small image to use for rendering and collisions. When initiated, each obstacle should apply a random rotation to the image that it is given to make the game more visually interesting. I will try assigning each obstacle a slightly different velocity but this could make the game too confusing so I will need to test this, I may choose to disable it on lower difficulty levels.

## Collisions

The sprite object in pygame has many useful methods so my obstacle class will be created as a child of the sprite class. This is especially useful for collisions as it includes the [sprite.collide\_mask(sprite1, sprite2)](https://www.pygame.org/docs/ref/sprite.html#pygame.sprite.collide_mask) method which lets me check for collisions by checking whether two bitmaps are overlapping. For each obstacle I will need to run [pygame.mask.from\_surface(surface)](https://www.pygame.org/docs/ref/mask.html#pygame.mask.from_surface) to make a mask that includes all the non-transparent pixels in the obstacle image. I will also need to get a mask for the player sprite. Then I can check collisions between the player and each asteroid. If this is too slow I may need to restrict the check to only include obstacles that are far enough down the screen to hit the player.

## Overview of class

### Init

* Get methods from parent sprite class
* Assign random position along a line above the top of the screen
* Choose velocity
* Load image with transparency
* Apply rotation to image
* Get mask for image

### Update

* Move position based on velocity, apply multiplier if player is moving extra fast

### Blit

* Draw onto the surface given

|  |  |  |
| --- | --- | --- |
| **Test** | **Expected result** | **Pass** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |