Jonathan Wilson

A-Level Programming Project OCR

Candidate number: 2251

Centre number: 48339

Analysis

Problem Identification

The restaurant I work at called Tekkerz would like to create some promotional material to advertise their new branch in Acomb. They would like something mainly focused on the younger generation and which might spread the promotional material organically. They also want to incorporate special offers into this material to encourage people to visit the new branch.

To solve this problem, I will create a promotional game to advertise special offers and their new branch in Acomb. I will create a game called Ghost of Acomb Tekkerz (a.k.a. GOAT). It will be a platformer with multiple levels, saved user progression and occasional interjections of promotional content. The user will start at one position in a and jump to different platforms and avoid obstacles to finish a level. It is inspired by games with platforming elements such as the Celeste and Hollow Knight.

- This will include a login and account creation screen where the user will enter their username and password in order to load their progression which will be saved at the end of a playing session.
- And a menu system that allows the user to set options and select a level to play.
- In the main game, there will be movement and jumping and continuous collision detection between the platforms and the player's hitbox.
- The player may also be able to unlock abilities like double jump or wall sliding/jumping.
- I may also implement different types of game objects the player can interact with, for example kill boxes (like lava) and bouncy platforms. which will be displayed to the user in a different colour and/or have different sprites so that the user can differentiate them from normal platforms. These would all make the game more engaging, interesting and fun, so the user will play for longer and see more promotional content for the Tekkerz chain of restaurants.
- And local multiplayer, where two users will play on the same keyboard, controlling two separate players (characters) and could compete or co-operate to complete a level.

This problem can be solved using the following computational methods:

Thinking Abstractly

This is the process of simplifying a problem or a concept, ideally keeping the simple and most important aspects and omitting the complicated and unnecessary parts. This is done to allow the solution to be created in reasonable time and possibly to make the solution better by making it clearer and simpler.

I will use abstraction by representing the player as a position and velocity, with the movement keys applying a constant acceleration. This will make it easier to use basic physics to control the player's movement.

The player will also have a rectangular hitbox, allowing the character to collide with rectangular platforms instead of the player's sprite. This is because collision with the sprite of the player would be very difficult to achieve and the player will probably not notice that the player is actually colliding with a hitbox.

I will also abstract people into a username and password along with the player's progression because that is all the program needs to save and load user data and for it to be easily accessible to the user.

The player's animations will also be abstracted into a few frames with different frames for different movements because fully animating the player is outside the scope of the game.

Thinking Ahead

This technique involves considering the inputs and outputs of the program and what will be used to create a solution.

I am planning to use the python library pygame to make my game because it is fairly simple, has all the features I need such as drawing shapes and images to the screen and managing inputs at the low level.

The inputs to the program will include:

- Text entries and buttons similar to HTML forms which take information so that the user can login and/or create an account so that the information can be checked against the list of accounts.
- Key presses so that the user can control the movement of the player.
- Mouse clicks and the position of the mouse so that the user can interact with the menu.

The outputs of the solution will include:

- Error messages and completion messages when the user enters a field in an entry/text. This will be done to notify the user when they have entered something incorrectly and tell them what they should do instead of leaving them clueless and annoyed.
- Saving user info into a file when the account is created so that the user can login next time and retrieve their saved progression.
- Drawing rectangles and sprites on the screen where the player and rectangles are so that the user can see where the player is and decide the next action they should take.

- Saving progression and level data to a file so the user does not have to repeat everything they have done so far to get back to where they left off.
- Displaying buttons and menus so the user can interact with them and (for example) choose a level.

Thinking Procedurally and Decomposition

This is where a problem is broken down into smaller more manageable problems which can be solved quickly in turn. I will do this so I don't get bogged down in trying to make everything at once and so I can make each individual component at a high quality. Issues can arise when integrating all the different components.

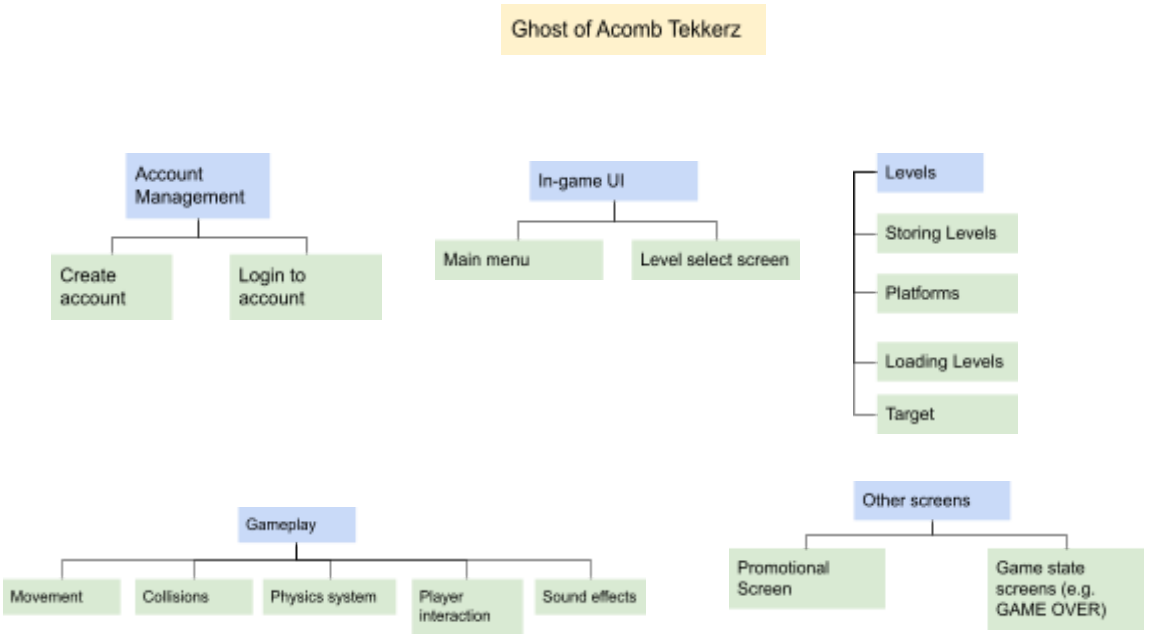
During the development of this game, I will apply decomposition by developing prototypes for different parts of the game, for example:

The player's movement, drawing a rectangle on the screen that moves left and right. Because it will be easier to develop this in isolation to collision with other rectangles and the ground.

Rectangle collision, drawing a moving rectangle that is blocked by another rectangle. Because rectangle collision is simpler when all the rectangles are moving at constant velocity with no acceleration from the movement of the player.

A menu system that allows the user to navigate the different screens with only a few buttons and limited utility. Because coding the menu system with lots of interaction initially makes it very complex, but only having one screen would not allow me to test the system.

The problem may be broken down as follows:



Thinking Logically

This is determining where to use selection and iteration in the solution. This will help me plan out how to solve different parts of the problem using these techniques.

In my game, I will use selection when:

- Deciding if the password created for a new account is valid so that passwords can't be guessed as easily.
- Determining if a key is pressed so a command can be sent to the player so that it moves.
- Determining whether the player is colliding with a platform e.g. "Is the player's character hitbox intersecting the rectangle of the platform?". This is done so that an appropriate collision response can be triggered, like moving the player out of the platform and changing its movement in some way.
- When the player clicks when interacting with the menu screen, an "if" statement will be used to check if the user's mouse is inside a button, so they can be taken to another section of the menu or a level can be loaded.

And I will use iteration when:

- Checking through all the current user profiles during account creation so a username isn't duplicated. This is important because no two accounts should have the same username as that is what identifies them.
- Logging in. The program will check all the account's usernames to see if one matches the username entered by the user.
- Drawing all the platforms because the platforms will be stored in a list. This will need to be looped over to carry out processes on the platforms.
- Similarly, when checking the player for collisions with the platforms, because all the platforms need to be checked once per frame for collisions, so the platform list will need to be looped over.
- During the menu screen, all the buttons need to be checked for clicks each frame. This will make the buttons responsive and not have any delay when interacting with them.

Thinking Concurrently

This is where two programs are running at the same time, either parallelised on different threads or cores or giving the illusion of running simultaneously by time slices being given to each problem. This must be implemented to allow multiple things to happen at the same time on the game screen. It can also speed up processes if they can be split up and carries out on multiple cores at the same time.

In the main game, this will be implemented using a game loop where the player is moved, everything else is updated (including collisions) and everything is drawn once per frame, this

will happen around sixty times per second (hopefully). Because it must look like it is all happening at the same time from the user's perspective.

Conclusion

In conclusion, this problem is amenable to computational thinking and computational methods because a simple but entertaining computer game is a great way to gain the interest of the younger generations.

This solution requires a computer program to solve because a computer is good at solving problems involving lots of calculation such as collision detection and physics simulation (for the movement of the player). This is beneficial because a computer's proficiency at this will mean that the program will run smoothly at a relatively high frame rate.

A computer can also store data in non-volatile storage like in an SSD or HDD, which means it is ideal for storing user, progression and level data. The computer can also load and save this data quickly and interpret it i.e convert it from binary to floating point numbers and strings.

The computational methods are also applicable to this solution because:

- Thinking abstractly can simplify the problem into its core components, making the solution simpler, quicker to program and easier for the user to understand.
- Thinking ahead allows me to plan the requirements of the project and the inputs and outputs, which will describe how the user will interact with the program and what information the program would give to the user.
- Thinking procedurally and decomposing will allow me to break down the problem into easier and quicker to solve sub-problems.
- Thinking logically will allow me to plan out how the problem will be solved and where techniques such as selection and iteration will be used so I can determine which parts of the solution will be more complex to design and program.
- Thinking concurrently will allow me to think about how different parts of the program will be executed at the same time, or give an impression of this happening to the user. This will mainly be achieved using a game loop, similar to the use of time slices in processor scheduling.

Stakeholders

Stakeholders are the people which my solution will affect. This can be because they benefit from the game being played, such as the owner of the Tekkerz restaurant chain, or because they are directly affected by the program in another way, such as being a user of the game, or customer at the restaurant. It is important to consider the point of view of these people because the game must promote the restaurant effectively, and the game must be entertaining to play so it is not ignored by potential users.

The owner of the restaurant is only interested in how the game will promote the restaurant. He wants the game to include references to the restaurant and special offers. He has a basic understanding of video games and knows that the customers most likely to play the game are in the younger generation. He hopes that the brand will be imprinted on these impressionable children so that they will come back to the restaurant later in life, seeking any nostalgic reminder of their childhood. He is playing the long game. My game will help achieve this goal by giving children easy access to promotional material presented in a fun and harmless way.

A typical user would be my brother. He has a fairly advanced knowledge of video games, as he has played some of the games that mine is inspired by. He is an appropriate stakeholder as he is in the target demographic as specified by the owner of Tekkerz. However, the target user does not necessarily have good video game or computer skills, as the user interface should be intuitive and easy to use. The game nust be fun to play because he and others in this demographic have so many other options for entertainment. This means that a balance must be struck between promotional material and quality of uninterrupted gameplay.

The target audience would be a 9-14 year old boy because they would be entertained by a relatively simple game and most people who play games and/or have past experience with games are male.

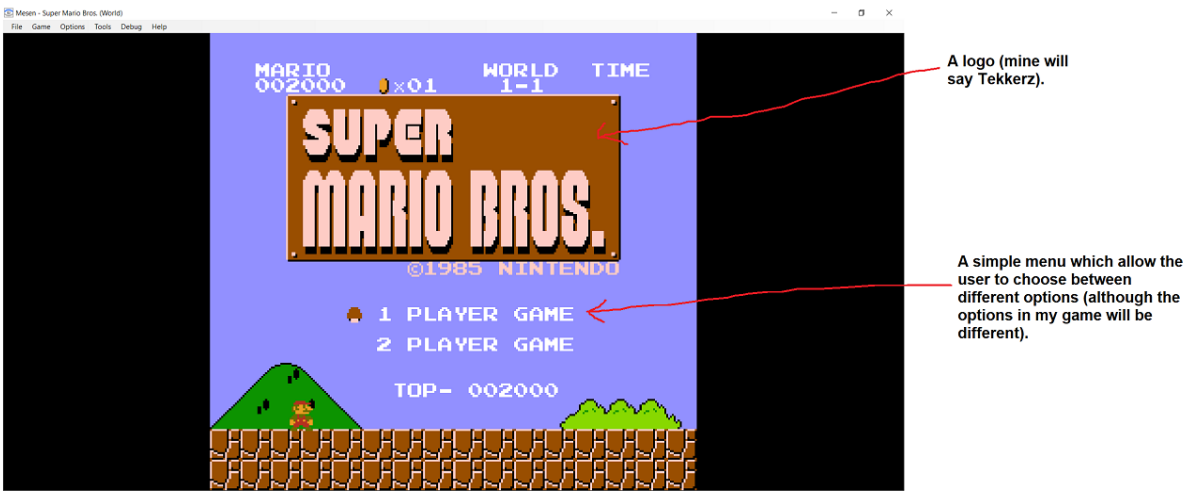
The owner of the restaurant cannot be easily contacted, so I will consult my brother on the quality of gameplay and try to occasionally contact the owner to ensure that there is sufficient promotional material.

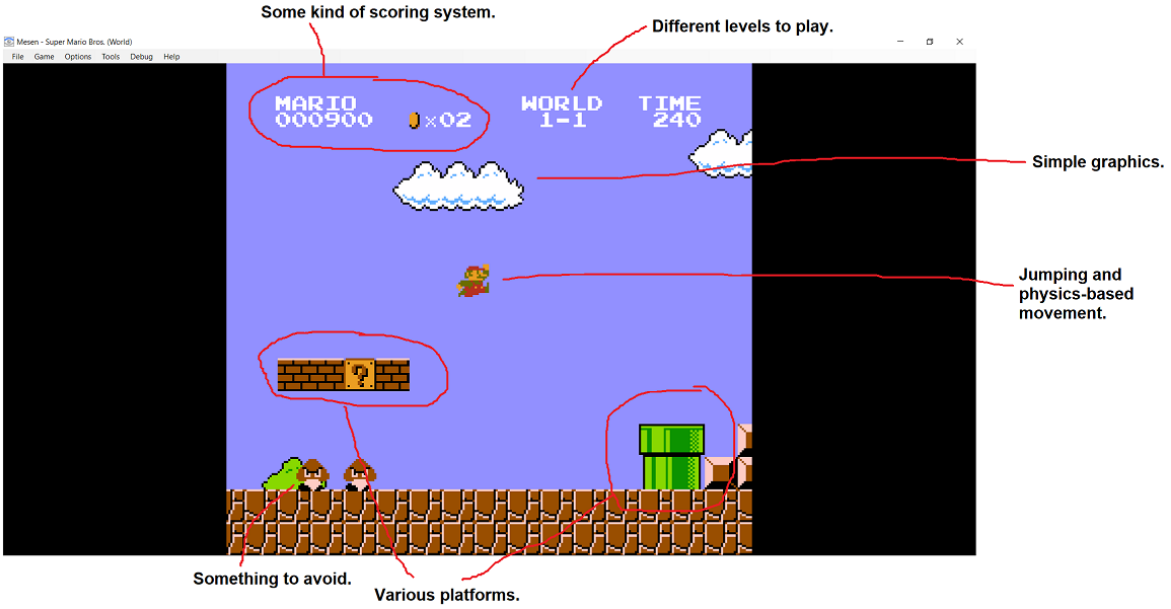
Research

Existing Solutions

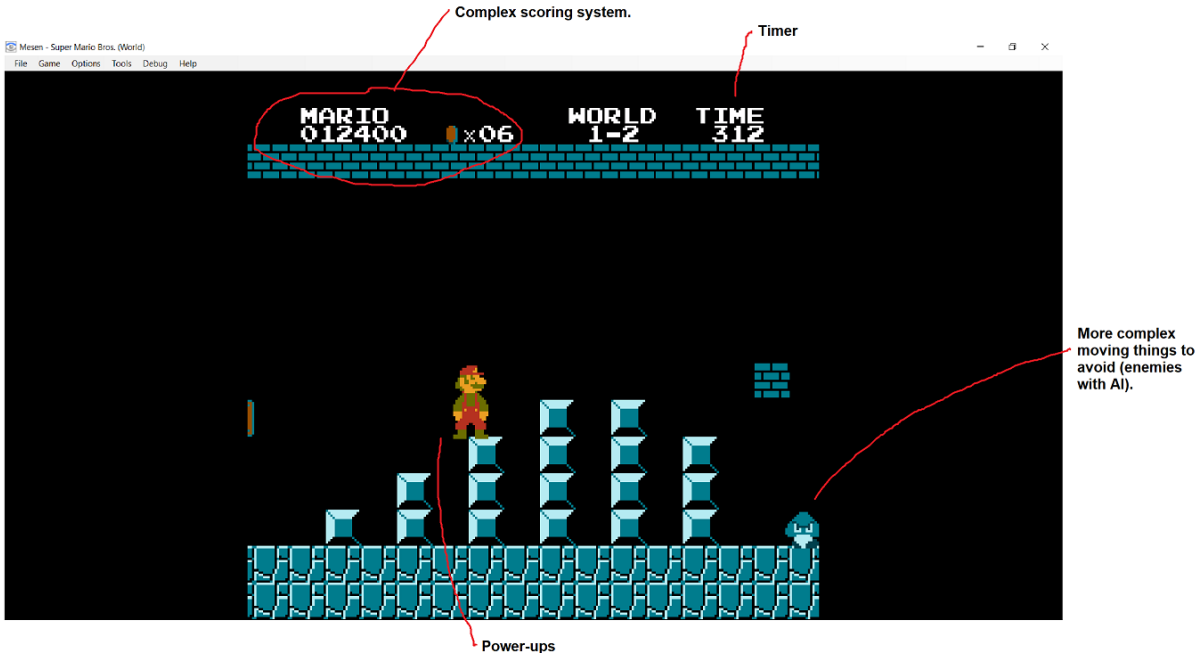
One existing solution is Super Mario Bros World. I will research this game so that I can gain inspiration for features and see what kind of solution is achievable. In this game, you complete different levels, avoiding obstacles to try and reach the goal. The graphics are simple, which will be similar to my game because I do not have the skills or time to create complex graphics. It also has platforming elements and a simple menu screen.

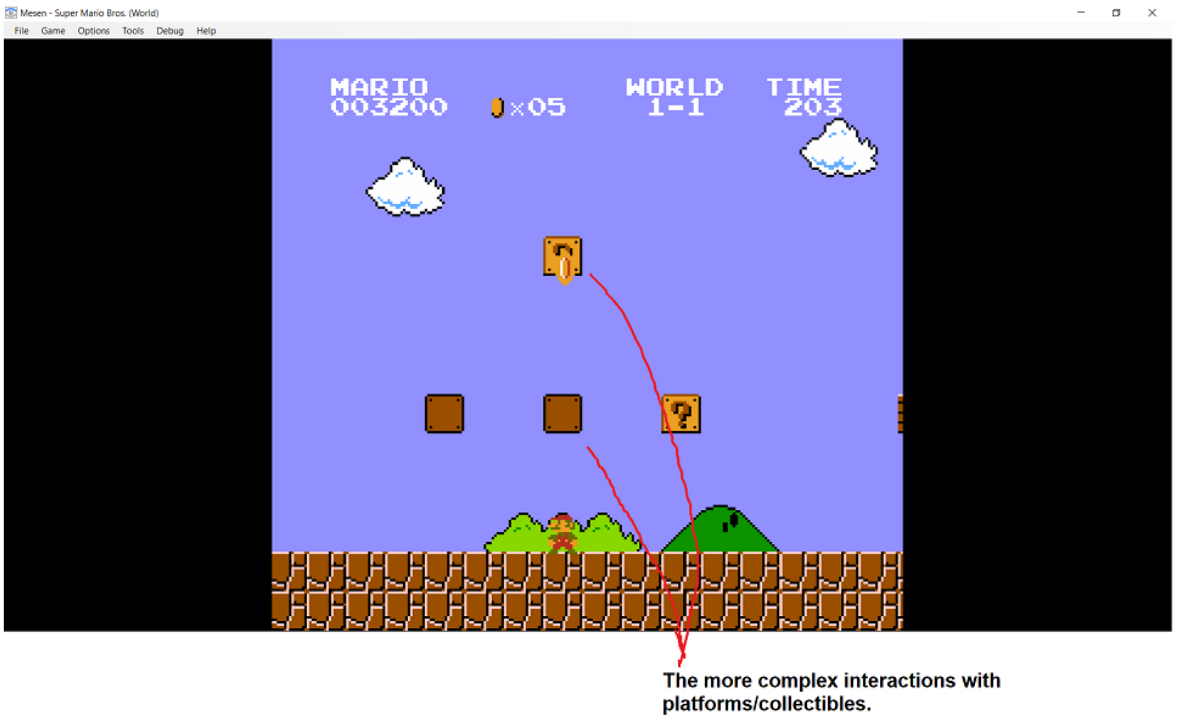
Some of the features from this game which I will include:





And here are some features from this game which I do not currently plan to include:

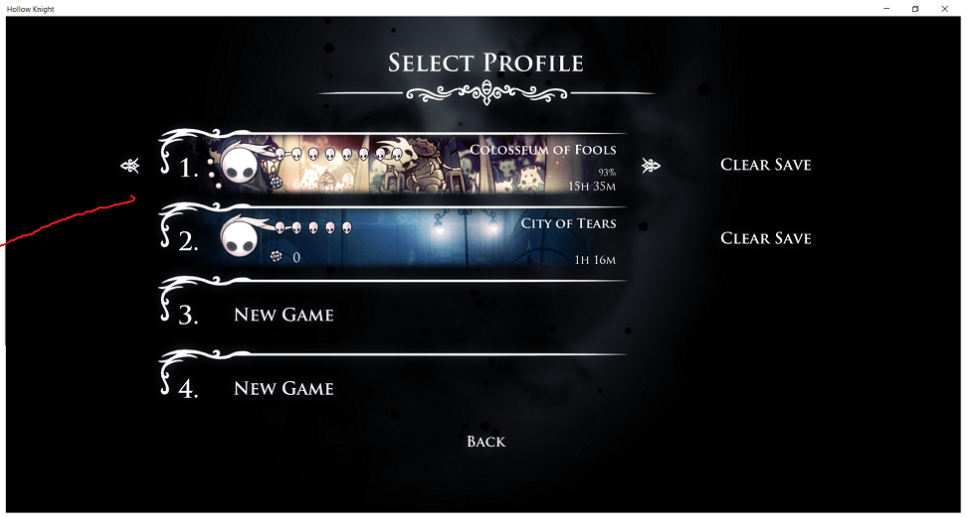




Feature	Do I plan to include this?	About the feature, and advantages and limitations of the feature.
Logo	Yes	The feature appears at the start of Super Mario Bros., and shows the name of the publisher (Nintendo). I plan to add this to my game to promote Tekkerz at the risk of detracting from the game itself.
Menu	Yes	This is part of the graphic user interface, Mine will be more complex than the one in this game, perhaps with an options screen and a level select screen. I also do not plan to implement the “2 PLAYER GAME” option because it would take a lot of time, and my game is intended to be single player anyway.
Physics-based movement and jumping	Yes	This makes the movement feel realistic and responsive. If implementing collision, it also means that levels can be designed with any arrangement of platforms, and all the movement will still work, as opposed to hard coding in ways that the player is supposed to move. This will allow me to speed up the development of the game in the long run, and therefore add more features in a given time frame.
Moving enemies	No	This will take a lot of time to get working correctly as each enemy has to follow its own specific path. Hopefully, my game will be challenging enough without these.
Interactions with platforms (the mystery boxes).	No	This feature is not really relevant to my game as the goal will be mainly to complete each level, the score will not be a priority. This is because at the end of each level, there will be a special offer at Tekkerz, so the more levels are completed, the more promotional material is viewed.

Another game that I will take inspiration from is Hollow Knight, although platforming is not the main aspect of this game, it implements a saving system which I can take inspiration from, and aspect of the game's movement could be implemented as well.

Some of the features from this game which I will include:



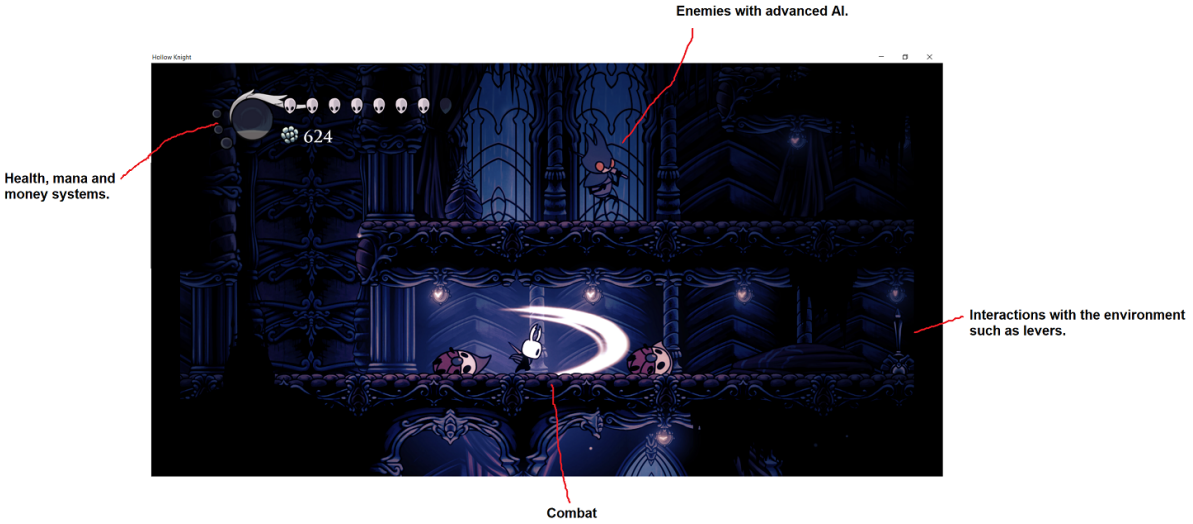
A system for saving progress and creating new saves.

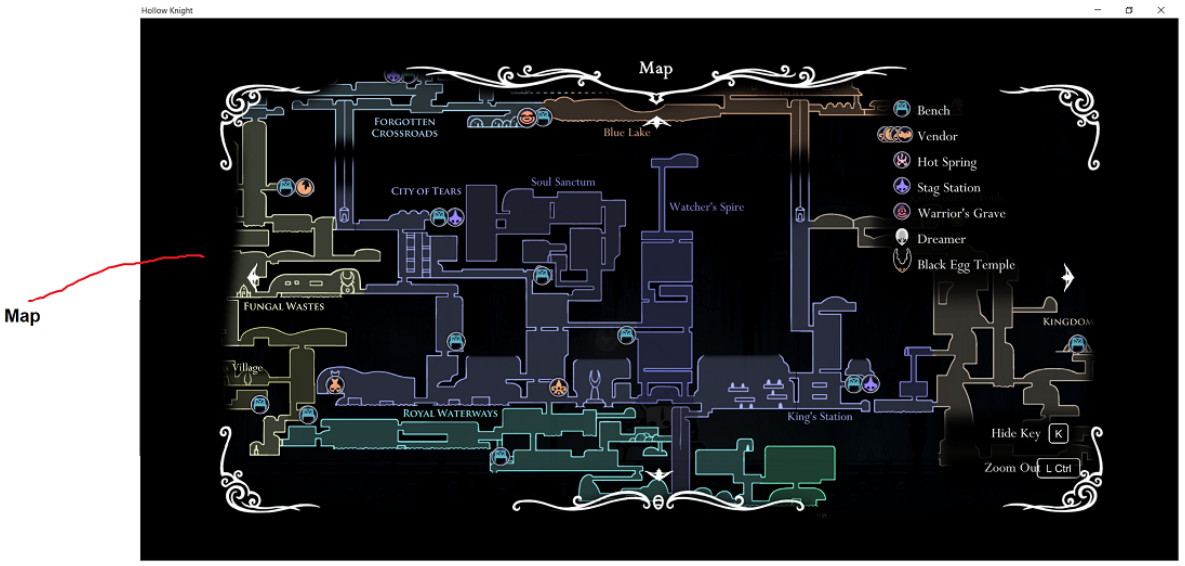
Special abilities such as double jump (as shown here), dash and wall climb.

Jumping and physics-based movement.

Kill-box (in this case acid) which will kill the player (or take off health, as in this case) if touched.

Some of the features from this game which I do not currently plan to include:





Feature	Do I plan to include this?	About the feature, and advantages and limitations of the feature.
A system for saving progression and creating new saves.	Yes	This feature will allow the user to begin where they last left off. In Hollow Knight, a user has a number of different saves and can create up to four saves in total. Similarly in my game, a save will be created by creating a new account, and each account will have their progress saved and can begin where they left off. This will be conducive to a more pleasant gaming experience because the user can take a break and come back, without having to complete the game all in one go.
Combat	No	My game will be strictly a platformer, so will not have combat, the reason for this is that Hollow Knight has a much larger scope, and the developers had more time to make it.
Mana health and money systems.	No	My game has no use for mana and money, its scope is too small. I also plan to only give the player one life for a level, which could be considered health, but not the same as in Hollow Knight
Map	No	Each level will be small enough to easily navigate. No map will be needed.
Kill boxes	Yes	These are rectangles, which cause the player to die if they enter. This will allow me to implement a lose/game over condition, so the player can actually fail the level. I plan to implement this similarly to the acid lakes in Hollow Knight. As an area under the platforms which causes the player to die if they fall down.
Special Abilities	Yes	I plan to add abilities such as double jump and wall climb to my game as they will make the movement much more interesting. However, I will not add as large of a range of abilities as in Hollow Knight because I don't have the time to do so.

Interview with Stakeholder

Interview with client:

I will conduct an interview with the owner of Tekkerz, to identify what features would make this game effective promotional material, and any personal preferences he has about the game. He may also consider some aspects of the game which will keep the user engaged, so is not only concerned with the promotional side of the game. This is because the longer the user is engaged, the longer the user will be viewing adverts and special offers for Tekkerz. In this way, this game will be a balance between showing the user promotional material and keeping the user engaged and interested in the game long enough to see this material.

How often should the game show promotional material and/or special offers and what should be shown?

The game should show special offers at the end of each level to reinforce the idea that a meal at Tekkerz is a reward.

How many levels should this game have?

The game should have as many levels as possible to keep the user engaged and viewing the brand for as long as possible. So fifty.

How should this solution handle multiple users?

The solution should allow for multiple users, and store each of their progressions, so that one copy of the game can reach as many people as possible.

If multiple users are allowed, how should they login?

There should be a simple and intuitive user interface to create an account and login. It should be simple so as not to discourage a younger user.

How challenging should this game be?

The game should have a moderate amount of challenge because a game with no challenge will not keep the user engaged. However, the game should not be extremely difficult because the target audience is the younger generation who may not be experienced at games like this.

What kinds of sounds should the game have?

There should be a reward/achievement sound when a level is completed to notify the user that they have received a special offer. There could also be sounds as the player moves, such as when they jump to make the game seem more responsive and engaging.

<u>Interview with example target customer:</u>

I will also conduct an interview with my brother Alex, who is a member of the target demographic of the game. I will interview him to find out what will make the game interesting for people like him.

Should this be a single player or multiplayer game?

It would be fun to play the game with other people, so multiplayer is ideal. Online multiplayer would also be good because I would like to play the game with my friends online.

What kind of graphics would this game have?

The graphics don't really matter as long as the game is fun. Particle effects and colourful animations can make games more interesting.

What kind of movement makes a platformer fun to play?

The movement should be fast and responsive.

What kind of special abilities should the player have in a platformer?

From my experience, double-jumping, wall-climbing and dashing make platformers fun.

How would I make any user interfaces intuitive for you?

Any buttons or text entries should be clearly labelled with what they do or what screen they take you to. The options the UI gives me should also be simple so it's not confusing.

What kind of sounds would the game have?

Background sound tracks that respond to the user's progression, for example music that builds up the further the player is in a level can make games more interesting.

Review of Interviews:

From these interviews, I intended to gain a better idea of what features the game should have to benefit the stakeholders, and what features would make the game more fun for someone who is a member of the target audience.

I learnt that the gameplay is more important than visuals, but that the game should have some visual effects and colourful sprites to make it visually appealing and not be perceived as boring by the target audience. However, I will have to limit the more advanced graphics because particle effects are beyond the scope of this game as it would take me too long to learn how to implement them.

Importantly, the game should have responsive and varied movement, for example including a double jump and wall climbing. This is because it would differentiate the different levels, as just horizontal movement and jumping may not be enough to keep the user engaged for multiple levels.

The user interface should be simple and easy to use as this is not the main part of the game. The user should not have to think about how to use it as it should be intuitive. This is

because a complex user interface may put a user off playing the game, especially if it is annoying to use.

A stakeholder (the owner of the restaurant) mentioned multiplayer. This would indeed make the game more fun, as competition between players always makes games more engaging. Although local multiplayer may be feasible, in general, multiplayer (especially online multiplayer) is beyond the scope of the solution because the interaction between the two players (two moving rectangles colliding) and networking over the internet when implementing online multiplayer would take too long to learn how to implement.

As for sound, multiple tracks of background music is beyond my skill set, and even if I had the ability to make a soundtrack, it would take too long. However, simple retro-themed sounds when doing different movements (for example jumping) would be well within the scope of my solution and would make the game feel more responsive.

Survey

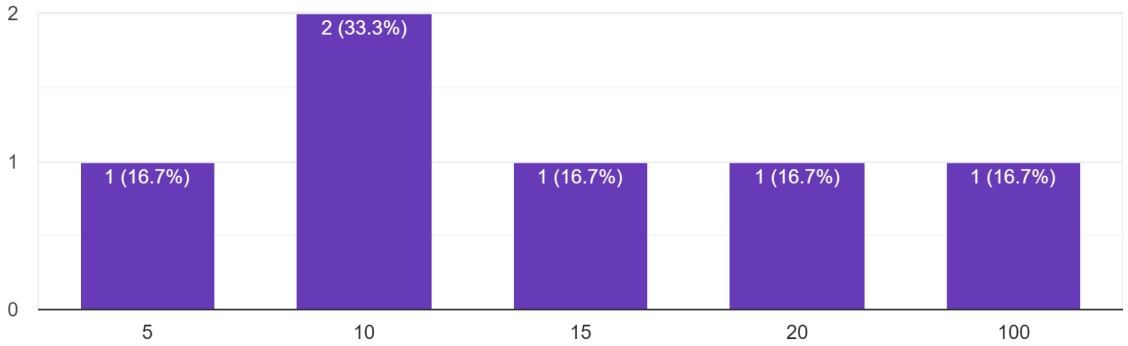
Although I have interviewed an individual from the target audience, I want to get a broader view of what the target audience wants from the game. To do this, I will create a survey asking people quantitative questions about features this solution could have. I will use google forms to create and distribute the survey.

I asked people who would be members of the target audience because I want to get an idea of what they would find engaging in the game. The stakeholder cannot provide this information because he is not a member of the target audience, but it is very important for the delivery and sharing of the promotional material. This means I must survey the target audience.

These are the questions and results from the survey:

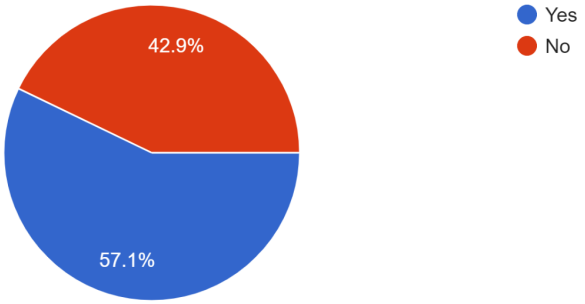
How many levels should this game have? (number)

6 responses



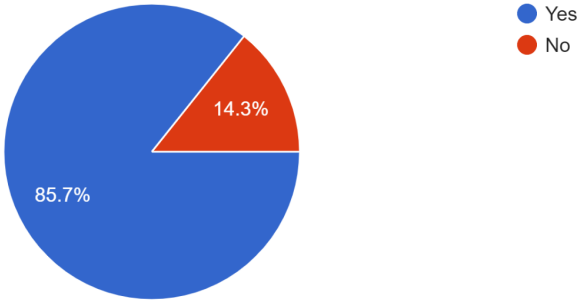
Should this game have multiplayer option?

7 responses



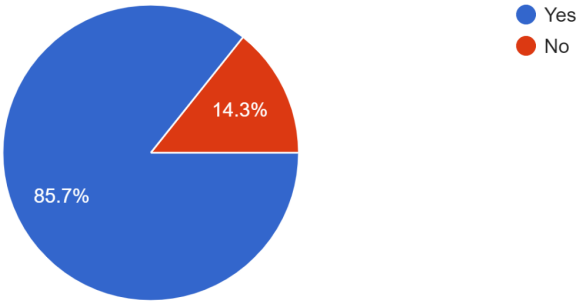
Should this game have double jumping and wall climbing?

7 responses



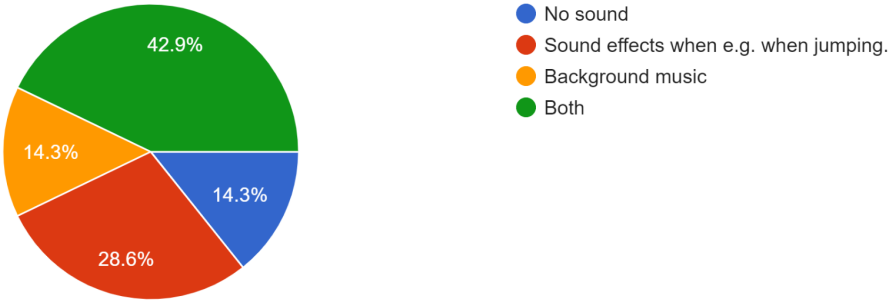
Should this game allow multiple users (with a login system)?

7 responses



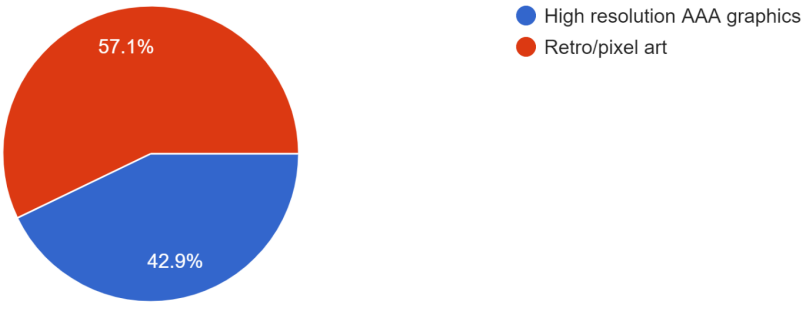
What kind of sounds should this game have?

7 responses



What kind of visuals should this game have?

7 responses



From this survey, I have gathered that most people want lots of levels to play. This would be ideal because it would keep the user engaged for longer, however, I don't think 20 levels is feasible because that much content would take too long to create.

Most people don't see the value of adding multiplayer to a platforming game, which are traditionally singleplayer. I won't be adding multiplayer to this game even though a large proportion of people suggested it because I do not have the time during the creation of the solution to learn how to implement this.

The vast majority of people wanted varied movement such as double jumps in the game. I will implement this because it would be fairly simple to implement once the general physics system has been created.

A lot of people wanted both background music and sound effects, but it is clear that the sound effects are most important to most people, so I will only add that to the game. Background music will take too long to create.

Also, six out of seven people wanted to be able to save progression on multiple accounts as different users, this makes sense because people want to be able to share their game with other people that use their computer. It would also be great for the stakeholder (the owner of the restaurant) because it allows the promotional material to be seen by multiple people for one distribution of the game.

Most people also preferred retro visuals over advanced AAA graphics, because they wouldn't really be appropriate in this simplistic style of game.

Proposed Solution

Essential Features

Number	Feature	Description	Justification	Evidence
1	Physics-based movement	Movement that incorporates applying forces to game objects such as to the player when movement keys are pressed and friction when interacting with platforms. This causes acceleration or deceleration of the player.	This will allow the movement to feel more responsive. It will also make the movement more intuitive because it will be closer to realistic movement.	Interview with example customer
2	Collisions	Interactions between the player	This is necessary for any platformer, as without	Other games

		and platforms, which allow the player not to fall through platforms.	collisions there are no functioning platforms.	
3	Levels	Multiple different sections of the game to play.	This will keep the user engaged and give them a few smaller goals, which makes the game seem more rewarding.	Survey of target audience
4	Saved progressio n	Saving a file containing information about how far the user is through different levels to secondary storage.	This will allow the user to pick back up where they left off. Someone may get frustrated if they have to start the game over every time they want to play, so this feature will keep the user engaged for more playing sessions.	Interview with stakeholder
5	User profiles	Different progression profiles for different users.	This will allow multiple users to access the game and have different saves of progression so each user can progress through the levels at their own pace.	Interview with stakeholder
6	Account screens	User interfaces for creating and logging into accounts.	This will make the user's interaction with their account simple and intuitive, as opposed to a text or command line interface, which could confuse a new user and possibly put them off the game entirely.	Interview with example customer
7	Player character is a ghost	The sprite for the game character is a retro-themed ghost.	This will provide an interesting character to keep the user engaged.	
8	Kill boxes	These are rectangles that kill the player if it collides with this.	This provides a fail condition so the player can actually fail the level. These can also serve as the level's bounding box so that the player can't fall into nothingness forever.	Other games
9	Menus	In-game menus (different from account screens). This includes a main menu screen, a level select screen and possibly	This will allow the user to intuitively interact with the game and select different levels. The main menu screen will allow the user to navigate to the other screen (such as the level select	Other games and interviews

		an options screen (if time allows).	screen) and exit the game.	
10	Double jump	A special ability that the player may be able to collect or have from the start. This will allow the player to jump again whilst being in the air.	This will make the game more fun to play so will keep the user engaged for longer so they will see more promotional material. It will also mean that the level can be made more complex, which will make the game more interesting.	Interview with example customer
11	Retro/ simplistic graphics	Graphics that use simple shapes or pixelated sprites.	This will give the game pleasing visuals without me having to spend too much time on the design on the sprites. This will allow me to focus on different aspects of the gameplay.	Survey of the target audience
12	“Game Over” screen	A screen that displays game over when the player fail a level	This will show the player that they have failed on a level and allow them to try again, which may keep the user engaged for longer.	

Optional (time dependent) Features

Number	Feature	Description	Justification	Evidence (if applicable)
1	Other special abilities	Other than a double jump, I may add special abilities such as wall climbing and dashing.	As with the double jump, this will make the game more engaging.	Interview with example customer and other games
2	A level editor	A simple user interface program which helps me create levels.	This will make the creation of new levels faster. This will help me get to the number of levels some of the target audience expected.	Survey of the target audience
3	Score	Some kind of score, based on collectibles.	This will give the user a target, other than completing the level. This will give them more to do, so they might spend more time playing the game.	Other games

4	Timer	This would not be a countdown timer like in Super Mario Bros., this would be a count up timer.	This would show the user how long it took them to complete the level. It could also make the levels more replayable.	
5	Other types of platforms	Bouncy or ice platforms.	These would make the game more interesting to play, and create more variety in the levels. It would also not be too time consuming to implement given a physics-based collision model.	Other games

Limitations

In my solution, it would not be feasible to include advanced AAA graphics (which a large proportion of the target audience which I surveyed suggested) because they would take too long to create. Also, I would like my solution to run on as wide a range of hardware as possible, and this is not feasible using these graphics. Another option is to create background music (another suggestion of the surveyed target audience), but this would not be feasible because it would take too long to make, and I don't currently have the skills to do this.

As for multiplayer, it would not be feasible to implement this because the interaction between the two players (two moving rectangles with their own velocity and acceleration) is too complicated for me to learn how to achieve in a limited time frame. Furthermore, an online multiplayer mode would also be outside of my current skill set because I would have to consider factors such as networking and running the game on a server which both users can connect to.

Another limitation is that I can't create a large number of levels. In an interview with the stakeholder (the owner of the restaurant), it was suggested that the more levels I added, the better. While this is true, time limitations will limit the number of levels to around ten. This could be increased by me adding a simple level editor to speed up the creation of levels. However this may also not be feasible because of time constraints.

Requirements

Minimum hardware requirements to produce my solution:

Requirement	Justification
4GB of RAM	This is recommended to run multiple tabs in a modern

	browser such as Brave or Chrome along with Windows 10 and VS Code (an IDE made with electron, so has high memory requirements like a browser). I will also have to run, test and debug prototypes of the game, but this will require a small amount of RAM, mostly just to run the python interpreter.
>1GHz Intel 8th Gen (or equivalent) or later CPU	This will allow me to browse the internet quickly and run my code editor uninterrupted. This is also the recommended CPU for Windows 10. Testing my game will not require this amount of processing power, so this will be plenty for that use case.
128GB SSD	An SSD will allow me to load programs and boot up Windows 10 faster, so I can create the game in a reasonable time. Windows 10 only requires 16GB of disk space, but I will need extra space to store my game's data, such as levels and sprites, and to store programs such as an IDE, a browser and the python interpreter.
Any kind of integrated graphics or discrete graphics card.	None of the programs I will run are very graphically intensive, so any kind of graphics processing unit will do.
Mouse and keyboard	These are necessary to create the game's code.
Speakers or headphones	I need these to test and create the sound effects for the game.
At least a 720p monitor	I will need this to test the game and interact with the IDE and browser to research for the game. A 1080p monitor would be recommended.

<u>Minimum hardware requirements to use my solution:</u>

Requirement	Justification
2GB RAM	The user will need to run Windows 10, which is what most of this memory requirement will be used for. This will also be used for the working memory for this game, storing the sprites and information about the current level and the player.
32GB HDD (or SSD)	Windows 10 requires 16GB of secondary storage, and the solution will need some storage space to store level and user data. The python interpreter also needs some secondary storage to operate.
Any kind of integrated or discrete GPU.	As the game will use simplistic/retro graphics, no advanced or discrete graphics card will be needed. Integrated graphics are suitable for displaying simple shapes and sprites.
>1 GHz CPU	This will be required to run Windows 10, and the python

	interpreter. A very small amount of processing power will be required to actually run the game.
Keyboard	This will be used to move the player and make the player jump. Also, the user will need to use a keyboard to enter their username and password to login and create their account.
Mouse	This will be used to interact with the account user interface and the game menus (e.g. the main menu and the levels screen).
Speaker or headphones	These will be required to hear the sound effects. Although these are not technically required, they are recommended (without them, the user will not be able to hear the sound effects).
At least a 720p monitor	The retro-style graphics do not need a high resolution to be fully appreciated. This resolution will allow all menus and user interfaces to be seen clearly enough.

Minimum software requirements to create solution:

Requirement	Justification
Windows 10	This will have all the features that my solution requires such as a window-based GUI and the ability to save, create and edit files. Most software is made for this platform, so I will be able to run any IDE I want, and browsers and python.
Python interpreter	I need this to run, test and debug the game and prototypes of the game.
Modern browser (e.g. Chrome or Brave)	This will help me research techniques to solve the problem on the internet. I will also use an online tool to create some pixel art.
VS Code	This is an IDE with lots of helpful features such as syntax highlighting, an integrated terminal in which to run my code, a debugger and the ability to expand its functionality with extensions (e.g. the python extension created by Mcrosoft which will help me run python files and give me explanations of the library functions and built-in functions that I will use).
Image editor such as the GNU Image Manipulation Program	This will allow me to create sprites for the game.

Minimum software requirements to use the solution:

Requirement	Justification

Any modern operating system.	This will provide a platform for the game to run on. It will manage the game's memory and provide a file management system to store and load the game program files and game data files (such as level files and local user data files).
A python interpreter 3.10.0 or later.	This can interpret all the latest python features that I will use, including lambda functions and list comprehension
Pygame	This is a python library that provides functionality to draw simple shapes and images to the screen and create windows.
Tkinter	This is a python library that I will use to create a login screen with buttons and text inputs. I don't need to create any complex UI as I want the login screen to be simple and intuitive. Tkinter is appropriate for this purpose.
Graphics drivers	This will allow pygame to draw shapes and images to the screen.

Success Criteria

Number	Requirement	Justification	Measurement Method
1	Runs at a suitable framerate.	This will make the movement responsive and provide a more pleasant user experience.	Runs at a minimum of 30 frames per second.
2	User friendly user interface.	This will make sure the user does not get frustrated and quit the game.	Survey users/testers during final testing.
3	Feature rich user interface.	This will allow the user to interact with the user interface to do exactly what they want to do, whilst also being intuitive.	The user interface must include: ● A main menu ● A level select screen with level numbers ● A text entry input ● Buttons ● A pause menu
4	The user must be able to simply	This will provide a structure within which to	There must be a create account and login screen with

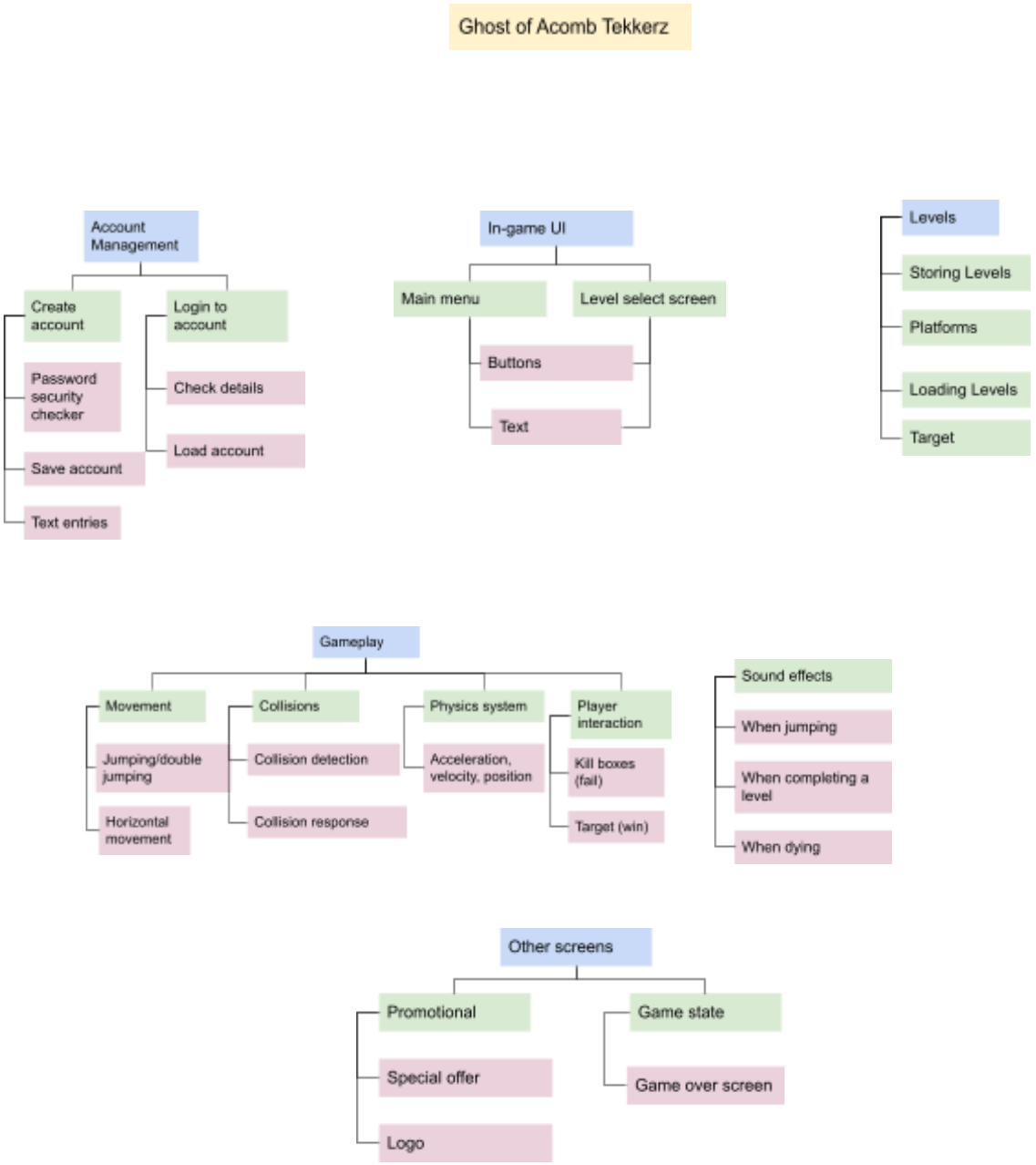
	create and log into an account.	store progression data. This means the user can resume the game where they left off.	a simple user interface. This simplicity will be quantified by surveying users/testers.
5	There must be sufficient promotional material for Tekkerz in the game.	This will promote the restaurant chain and fulfil the function desired by the stakeholder.	I will query the stakeholder. There must also be a special offer at the end of a level and the Tekkerz logo on the main menu.
6	The game must contain enough playable content to keep the user engaged for a reasonable amount of time.	This will keep the user playing the game to see promotional material.	I will survey testers/users and ask how long they were kept engaged. The game must also have at least ten levels that are loaded in from secondary storage.
7	The game must be visually appealing with simple retro graphics.	This will hook the user in initially.	Survey of testers/users. The game must display a player sprite and platforms.
8	The game's skill ceiling must be high.	This will mean that the user can improve a lot from beginning playing to the end of the game. This will keep the user engaged and possibly give the game some replayability.	The player must have some special abilities such as double jumping and the levels must get progressively more difficult. This will be quantified by surveying the testers/users and asking them how hard they found each level and whether they were able to complete it.
9	The game's movement must be responsive and versatile.	This means that the user will be kept engaged for longer and the game will be more interesting.	The game must have horizontal movement, jumping and must collide with platforms. I will survey the testers/users and ask them how responsive the game feels.
10	Information about the game's state must be shown to the user in a clear way.	The user must know what is going on in the game, or they will get confused and stop playing.	The game must have a game over screen and a level finish screen.
11	Each level must include a way to win and a way to fail.	This gives the user a target and something to avoid, which will motivate them to play the game.	Each level must include a target area which will trigger the level win state and kill boxes which will trigger the level fail state if collided with.

12	A user must be able to save the state of the game to secondary storage.	This will be implemented so that the user can load their game progress after turning their computer off. This will stop them from being annoyed from having to play completed levels again.	There must be a save button that will save the user's progress in the level. This should also happen on level completion.
13	Actions in the game must give appropriate feedback to the player.	This will make it apparent that the user's input has been processed and alert them to its effect on the game.	The must be visible movement when the player moves because of an input. Also, sound effects must be played when certain things happen in the game such as the player jumping. When surveying the testers/users, I will ask them if they feel that they receive feedback from the game when they interact with it.

Design

Decomposition and Structure

To plan out the development of my project, I will decompose the problem because this will help me structure the solution. I will split the problem of creating a game into sub-problems, and then each of those into smaller sub-problems. This will also help my code be more modular and therefore be more maintainable.



Password security checker:
I will check that the password the user enters when they create their account fulfils some criteria such as being 5 or more letters long and contains a special or upper case character. I will add this because I don't want anyone logging into someone else's account and ruining their progress by guessing a password.

Save account:
The program will take the user's information and save it to a file so that the user can log in later.

Text entries:
These are user interface elements that allow the user to input their information. This will allow the program to create the account with that information. It is also a ubiquitous way of handling string inputs, so should be intuitive for most users.

Check details:
When a user logs in, I will check the details they have entered against data stored in the program. Then if the details (username and password) match, then I will load their progression and allow them to play the games that user.

Load account:
This refers to loading the accounts from secondary storage into working memory. This will be done so that the accounts can be saved in non-volatile storage so that people can shut down the computer before playing again. They can then be loaded into main memory so they can be quickly accessed and changed during the gaming session.

Main menu:
When the user first logs in, the game will show a main menu, so the user can choose between some options such as a level select screen and possibly an options menu.

Level select screen:
This is a screen that will show a list of levels as buttons including the level number This will allow the user to see which levels they have completed and their progress in the levels.

Buttons:
These will be displayed as rectangles on the screen that contain text. Some code will be run when they are clicked. I have chosen to use buttons because they are so common, they should be an intuitive way for the user to interact with the game.

Text:
This will show the user what screen they are on and be used to label buttons. This will help the user know how they are interacting with the program and make the user interface more clear.

Storing Levels:
This will not be part of the code, but I will create a JSON file and edit it by hand to create the levels. I will store the level data in secondary storage. I will use a JSON file instead of a plain text or binary file because it will be human readable, so easier to edit later.

Platforms:
These are components of a level which are sold blocks which the player can stand on and be blocked by. I will add these because without them there would be no game. The player needs something to stand on to stop them falling into the bottomless void. There must also be obstacles to the player to give the game an element of challenge.

Loading Levels:
This will be when the program loads the level data from the levels file and deserialises it into python objects. This will be useful so that level data does not have to be hardcoded into the game. It will also allow me to more easily add levels to the game by editing the level data file instead of the project's code.

Target:
This will be the object in the level which marks the end of the level. If the user hits the target, then the level will be won and the win state will be triggered. I will add this because it gives the user something to aim for instead of wandering around randomly in the level. It will also give me the opportunity to give the user special offers at an appropriate point in the game, to promote Tekkerz.

Jumping/double jumping:
This will be an instantaneous impulse of movement upwards that the player can trigger using a key on the keyboard. This will add verticality to the game and allow the levels to be more complex, therefore increasing the skill ceiling.

Horizontal movement:
The player will be able to move from left to right by applying acceleration to the player when they press some keys on the keyboard. This will allow the user to traverse the level in a way similar to running.

Collision detection:
This is the process of the program checking for collisions between the player and other game objects. I will implement this so that the player can respond to collisions with platforms at an appropriate time, and to check if the player has completed the level. I will also be used to check whether the player is intersecting with a kill-box, and should therefore die. I will implement this because it allows the program to trigger a collision response when appropriate.

Collision response:
This is what happens after a collision is detected. In some cases, such as when the player is colliding with a platform, it will be moving the player such that they are no longer in collision. Or, it could be causing the player to die or trigger a win state if the player collides with a kill-box or the target respectively. I will implement this because without collision response, collision detection is pointless.

Acceleration, velocity, position:
These are all properties of the player, as it moves through the level. THey allow the player to move in a realistic, physics-based way. This will make the movement more responsive.

Kill boxes:
These will be areas in which the player will die on entering. I will add this because it will make that game more challenging by adding a fail condition, so the user can actually fail the level.

Target:
This will be an area in which the player will win the level on entering, triggering the win state. I will add this because it will give the user a defined goal, instead of having them jump around aimlessly.

Sound effect when jumping:
I will add this to make the game feel more responsive. This will also give the user validation that their input (pressing the jump key) has been processed.

Sound effect when completing the level:
This will notify the player of their victory along with the win state screen. It will also act as a reward.

Sound effect when dying:
This will notify the player of their failure along with the game over screen.

Special offer:
The user will receive a special offer for a meal at Acomb Tekkerz when they win a level. I will add this because the stakeholder (owner of the restaurant) has asked me to do so in the interview.

Logo:
This will act as further promotional material to make the user aware of the Tekkerz brand.

Game over screen:
This will trigger when the player fails a level, notifying them that they have some so. It will also allow them to retry the level.

Algorithms

To plan my solution more thoroughly, I will design a possible layout for the code I will use for each section in pseudocode. This is because it will help me structure my code when I write it in python. I will also explain how and why I have structured my pseudocode in these ways and how the code will work.

Password security checker:

This piece of code will create some new UI elements for the username and password input and submit button. Then the submit button is set to get the contents of the inputs and check the password for length and space characters, then only save the user and open the login screen if the password is adequate. It also checks the username entered to make sure it is unique because the username will be the unique identifier for an account.

I have structured the password security checker as part of the create account section of the game because this is functionality which will only be used when creating the account. I have split the pseudocode up into a function for checking the password and a procedure to describe the action of the button on click because then the resulting code is a pure function which is easily identifiable to check whether any password is adequate or not. This will also not be part of a large block of code which would make it more confusing to maintain. A button's on click code also has to be a procedure.

```
usernameInput = new TextInputGUIElement()
passwordInput = new TextInputGUIElement()

submitButton = new ButtonGUIElement()

function passwordSecurityCheck(password)
    if password.length() < 5 then
        return false
    endif
    if password.contains(" ") then
        return false
    endif
    return true
endfunction

function usernameCheck(username)
    if usernameAlreadyExists(username) then
        return false
    endif
    if username == "" then
```

```
        return false
    endif
    if username.contains(" ") then
        return false
    endif
    return true
endfunction

procedure submitButtonOnClick()
    username = usernameInput.getContents()
    password = passwordInput.getContents()

    if usernameCheck(username) and passwordSecurityCheck(password) then
        user = new User(username, password)
        saveUserToSecondaryStorage(user)
        closeCurrentUIScreen()
        openLoginScreen()
    endif
endfunction

submitButton.setOnClick(submitButtonOnClick)
```

Save account:

This section of code is a procedure which will load the users from a file into an array. Then it will add the new user to the array, serialise the array again and save it back to the file.

The save account code is part of the create account branch because the only time a new user will be saved to secondary storage will be when an account is created. I have separated this into its own function so that it is separated from the rest of the code, so is more maintainable. Also, it will be easier to potentially reuse and change this functionality in the future.

```
procedure saveUserToSecondaryStorage(user)
    file = open("users.txt")
    serialisedUserArray = file.read()
    userArray = deserialiseObject(serialisedUserArray)
    userArray.push(user)
    serialisedUserArray = serialiseObject(userArray)
    file.write(serialisedUserArray)
    file.close()
```

```
endprocedure
```

Text Entries:

This class will be a blueprint for an object which is a text input for a GUI. It will use the tkinter GUI library. This has a private set procedure for the contents of the input which will be used by the tkinter input object to change the contents attribute when the user types on the keyboard. There is also a get function for the content attribute so that other parts of the program can use this input.

I have placed this in the create account branch because this is a vital part of the create account UI screen, although it will also be used in the login screen. I have used a class for this because a text entry input can be expressed as an object which has an attribute (the contents of the text input). I also want to create multiple instances of the text input, so the blueprint aspect of a class is ideal for this use.

```
class TextInputGUIElement

    public procedure new()
        content = ""
        entry = new tkinter.Entry()
        entry.setOnChange(setContents)
    endprocedure

    private procedure setContents(newContents)
        contents = newContents
    endprocedure

    public function getContents()
        return content
    endprocedure

endclass
```

Check details:

This section of the program will create two inputs for the username and password and a button to submit. When the submit button is clicked, the on submit procedure will be run and the user with that username will be loaded from the file, and if the passwords match, then the game will start.

I have placed this in the login to account section because this is the logic that happens when the login screen is run. I have created the get user function because this is separate functionality which could be reused. I have created an on submit procedure because the submit button needs a procedure to change the behaviour on click.

```
usernameInput = new TextInputGUIElement()
passwordInput = new TextInputGUIElement()

submitButton = new ButtonGUIElement()

function getUserWithUsername(username)
    users = loadUsersFromSecondaryStorage()

    for i = 0 to users.length() - 1
        if users[i].getUsername() == username then
            return users[i]
        endif
    next i

    return false
endfunction

procedure submitButtonOnClick()
    username = usernameInput.getContents()
    password = passwordInput.getContents()

    user = getUserWithUsername(username)

    if user != false then
        if user.username == password then
            closeCurrentUIScreen()
            startGame(user)
        endif
    endif
endfunction

submitButton.setOnClick(submitButtonOnClick)
```

Load account:

This is a function which will load the users from a file in secondary storage called users.txt, deserialise them and then return them as an array.

I have chosen to make this part of the log in to account section because to check the usernames and passwords entered against the existing usernames and passwords, the users' data must be loaded from secondary storage into main memory. I have also chosen to separate this piece of code into its own function because it will make it more reusable and maintainable. Also, it will be easier to change this logic if the structure of the users' data changes than if this code was embedded in a much larger function or procedure.

```
function loadUsersFromSecondaryStorage()
    file = open("users.txt")
    data = file.read()
    users = deserialiseObject(data)
    return users
endfunction
```

Main menu:

This code will load the Tekkerz logo and display it on the screen using pygame. It will then create buttons for navigation to the level select screen and any other screens I may add in the future.

This is part of the in-game UI because I want a seamless transition between the different menus and the game. This is not so important for the transition between the login screen and the game. I am not using the tkinter buttons because to achieve this seamless transition, all of this must be in the pygame window, so I must create the buttons and text myself instead of using pre-built ones from the tkinter library.

```
logoImage = loadImage("tekkerzlogo.png")
pygame.drawImage(logoImage)

levelSelectScreenButton = new Button("Go to Level Select Screen")
levelSelectScreenButton.setOnClick(runLevelSelectScreen)

// this is just a placeholder for other buttons that may be on this
screen
optionsScreenButton = new Button("Go to Options Screen")
optionsScreenButton.setOnClick(runOptionsScreen)
```

Level select screen:

This section of the program will be contained within a procedure which is run when the user clicks the level select button on the main menu. This procedure will load the level data from a file called levels.txt into an array. It will then create a button for each level, creating text for that button using the level number. Then it will set the on click procedure of each button to a procedure that runs the level in question.

I have structured this as part of the in-game UI because it is a user interface that the user can interact with, and it will be part of the game window to achieve a seamless transition between the level select screen and the gameplay.

```
procedure runLevelSelectScreen()
    levels = deserialiseObject(loadLevels("levels.txt"))

    levelButtons = []

    for i = 0 to levels.length() - 1
        newLevelButton = Button("Level " + i.toString())

        procedure runCurrentLevel()
            runLevel(levels[i])
        endprocedure

        newLevelButton.setOnClick(runCurrentLevel)
        levelButtons.push(newLevelButton)
    next i

endprocedure
```

Buttons:

This is a class used as a blueprint for a button in the in-game UI. When it is created, it will find an available area of the screen to place the rectangle of the button, and create some text to label the functionality of the button, and its on click procedure will be set to an empty procedure. This class will also have a method to change what happens when the button is clicked and a procedure to draw the button and text label to the screen. There will also be an on click method which will be called whenever the user clicks the mouse and checks whether the button should be clicked or not.

I have placed this section of code in the in-game UI branch for the main menu and level select screen because these are two examples of the screens in which it will be used. It will also be essential for any other in-game UI which I might implement in the future. I cannot use the tkinter button for this purpose because the button must be part of the pygame window to achieve a seamless transition between the different UI screens and the game.

```
class Button

    public procedure new(text)
        rectangle = getAvailableRectangle()
        text = new Text(text, rectangle)

        procedure onClickProcedure()

        endprocedure

        onClick = onClickProcedure
    endprocedure

    public procedure click(mousePos)
        if pointInRectangle(rectangle, mousePos) then
            onClick()
        endif
    endprocedure

    public procedure draw()
        pygame.drawRectangle(rectangle)
        text.draw()
    endprocedure

    public procedure setOnClick(onClickProcedure)
        onClick = onClickProcedure
    endprocedure

endclass
```

Text:

This piece of code is a class which is a blueprint for a text label for an in-game UI screen. The justification of this is that it will show the user what different elements of the UI will do. When the text is created, the text and the rectangle that the text will occupy will be stored in the object and the size of the text will be calculated using a function that calculates how large text can be so that it will fill a rectangle. The only other method is a draw method which displays the text using pygame. There are no other methods as the text must be seen but has no interactivity.

This section of code in the in-game UI branch for the main menu and level select screen because it will be used for the in-game UI including these two screens and any other screens or parts of the UI I may implement in the future. I cannot use the tkinter text label UI element because this is for text in-game and in the in-game UI which uses pygame not tkinter.

```
class Text

    public procedure new(text, rectangle)
        text = text
        rectangle = rectangle
        textSize = getSizeOfTextToFillRectangle(text, rectangle)
    endprocedure

    public procedure draw()
        pygame.drawText(text, rectangle.x, rectangle.y, textSize)
    endprocedure

endclass
```

Storing Levels:

This section of the solution does not need pseudocode as this will be achieved by creating level data by hand and entering it into a JSON file in secondary storage. This can then be loaded later by the game.

I have placed this section in the levels branch of the program because it is part of the process of processing levels and manipulating secondary storage for saving levels.

Platforms:

This piece of code will be a blueprint for creating platforms, which will be rectangles. I will use a class here because it will be easier to create lots of platforms, encapsulating their data. The platform will mainly be used when loading levels from the JSON file. This will

contain a method to get the positions of the sides of the platform's rectangle so that it can more easily be used in collisions with the player. (These collisions will be done in the player class, not here.)

I have structured this section of code as part of the levels branch because it will be commonly used in the levels, and the platforms' data will be stored in secondary storage along with the levels' data.

```
class Platform

    public procedure new(x, y, hw, hh)
        rect = new Rectangle(x, y, hw, hh)
    endprocedure

    public function left()
        return x - hw
    endfunction

    public function top()
        return y - hh
    endfunction

    public function right()
        return x + hw
    endfunction

    public function bottom()
        return y + hh
    endfunction

endprocedure
```

Loading Levels:

This section of code will be a function that takes the path to a JSON file containing data for the levels. It will open the file and create a list of dictionaries from the JSON string in the file using the parseJSON function. It will create a Vector object for the start position and rectangles for the level bounds and target. It will also create an array of platforms in the level. It will then create a level object and store that in the levels array. I must do this conversion because JSON is human-readable so I can edit the levels in a text editor. And python objects are more robust and easier to work with than dictionaries.

I have placed this piece of code in the levels section of the tree because it handles all the data concerned with the levels. And this will be needed to load the data for the levels when the user enters the level select screen.

```
function load_levels(levelsFilePath)

    levelsFile = open(levelsFilePath)
    levelDicts = parseJSON(levelsFile.read())

    levels = []
    for levelDict in levelDicts

        startPos = new Vector(level_dict["start_pos"]["x"],
level_dict["start_pos"]["y"])

        target = new Rectangle(
            levelDict["target"]["x"],
            levelDict["target"]["y"],
            levelDict["target"]["hw"],
            levelDict["target"]["hh"],
        )

        bounds = new Rectangle(
            levelDict["bounds"]["x"],
            levelDict["bounds"]["y"],
            levelDict["bounds"]["hw"],
            levelDict["bounds"]["hh"],
        )

        platforms = []
        for platformDict in levelDict["platforms"]:
            platform = new Platform(
                platformDict["x"],
                platformDict["y"],
                platformDict["hw"],
                platformDict["hh"],
            )
            platforms.push(platform)
        next platformDict

        level = new Level(
```

```
            platformDict["level_id"],
            startPos,
            target,
            bounds,
            platforms,
        )
        levels.push(level)

    next levelDict

    return levels

endfunction
```

Target:

This is a component of the level. It is a rectangle, which on collision with the player will trigger the win state. This section does not require pseudocode as it is just a rectangle, using a generic rectangle class.

I have structured this section of code as part of the levels branch because it will be commonly used in the levels, and the target's data will be stored in secondary storage along with the levels' data.

Jumping/double jumping:

This section of code will check the user's input (specifically the jump input (spacebar)) and whether they are on the ground and apply an upwards acceleration to the player if all the criteria for jumping are met. It will use a method to check whether the player is on the ground. And a method to check whether the player still has their double jump.

I have placed this section of code in the movement branch because jumping is moving and will make the game more engaging. It is in the player class because it uses and controls data on a player object.

```
class Player

    ...

    public procedure jumpDoubleJump()
        if jumpKeyIsPressed() then
            if isTouchingGround() then
```

```
                accelerateUpwards(10)
            else hasDoubleJump() then
                accelerateUpwards(10)
                setCannotDoubleJump()
            endif
        endif
    endprocedure

    ...

endclass
```

Horizontal Movement:

This piece of code will take the user's inputs and convert them into an acceleration used to move the player.

I have placed this section in the gameplay movement branch because the player needs to be able to move to complete the level. Also, horizontal movement is player movement.

```
class Player

    ...

    public procedure horizontalMovement()
        xMovement = 0
        if leftIsPressed() then
            xMovement -= 1
        endif
        if rightIsPressed() then
            xMovement += 1
        endif
        applyHorizontalAcceleration(xMovement * speed)
    endprocedure

    ...

endclass
```

Collision Detection:

This section of code will detect a player intersecting with another rectangle, e.g. a platform or kill-box or target. It will check that one extremity of the player is outside the opposite extremity of the other rectangle and return false if so. It will do this for each side of the other rectangle.

I have placed this section of code in the player class because I plan to make the player object the only object in the game to collide with other game objects to make the game more simple, so it can be finished in reasonable time. I have placed it in the gameplay and collision sections, because it deals with collision and will only be run in the gameplay loop.

This code assumes that the position of a rectangle is in its centre and the size attribute is a vector pointing from its centre to the positive corner (usually bottom right).

```
class Player

    ...

    public function isCollidingWithRect(otherRect)
        if rect.x - rect.size.x >= otherRect.x + otherRect.size.x then
            return false
        endif
        if rect.y - rect.size.y >= otherRect.y + otherRect.size.y then
            return false
        endif
        if rect.x + rect.size.x <= otherRect.x - otherRect.size.x then
            return false
        endif
        if rect.y + rect.size.y <= otherRect.y + otherRect.size.y then
            return false
        endif
        return true
    endfunction

    ...

endclass
```

Collision Response:

This section of code assumes that the player is colliding with the other rectangle so must be preceded by a collision detection. It first simplifies the collision between two rectangles into the collision between a rectangle and a point. The algorithm will then check which edge of the rectangle is closest to that point and then move the player's rectangle outside of the rectangle on that edge (exactly on the edge of the rectangle doesn't count as collision).

I have placed this section of code in the player class because it will be used by the player to collide with platforms and kill-boxes, etc. I have placed this section in the gameplay and collision branch because collision response will only take place in the gameplay and it is part of the collision process.

```
class Player

    ...

    public procedure rectangleCollisionResponse(otherRect)

        bigRect = new Rectangle(
            (otherRect.x, otherRect.y),
            (otherRect.size.x + rect.size.x, otherRect.size.y +
rect.size.y)
        )
        point = new Vector(rect.x, rect.y)

        distToTop = point.y - bigRect.top()
        distToRight = bigRect.right() - point.x
        distToLeft = point.x - bigRect.left()
        distToBottom = bigRect.bottom() - point.y

        minDist = min(distToBottom, distToLeft, distToRight, distToTop)
        if minDist == distToBottom then
            rect.y = bigRect.bottom()
        else if minDist == distToLeft then
            rect.x = bigRect.left()
        else if minDist == distToRight then
            rect.x = bigRect.right()
        else if minDist == distToTop then
            rect.y = bigRect.top()
        endif

    procedure
```

```
    ...

endclass
```

Acceleration, velocity, position:

This section of code will use the acceleration to update the velocity and the velocity to update the position of the player. It will use Newton's physics equations to do this while being framerate independent.

```
class Player

    ...

    public procedure updatePhysics(deltaTime)
        position += velocity * deltaTime + 0.5 * acceleration *
deltaTime * deltaTime
        velocity += acceleration * deltaTime
    endprocedure

    ...

endclass
```

Kill boxes:

This section of code will handle the interaction between the player and the kill boxes. To do this, it will use a pre-existing method to check whether the player is colliding with the rectangle hitbox of the kill box and then trigger the fail state if it is.

I have used a method on the player class to check collisions because it is designed to be re-used in both the kill box collision and platform collision processes to detect collisions. I have placed this section of code in the game loop because it has to be able to trigger a change in game state. It does not really make sense to do this in the player class or kill box (rectangle) class.

```
procedure gameLoop()
```

```
    ...

    for killBox in killBoxes
        if player.rectangleCollisionDetection(killBox) then
            triggerFailState()
        endif
    next

    ...

endprocedure
```

Target:

This section of code will handle the interaction between the player and the target. To do this, it will use a pre-existing method to check whether the player is colliding with the rectangle hitbox of the target and then trigger the win state if it is.

I have used a method on the player class to check collisions because it is designed to be re-used in both the target collision and platform collision processes to detect collisions. I have placed this section of code in the game loop because it has to be able to trigger a change in game state. It does not really make sense to do this in the player class or target (rectangle) class.

```
procedure gameLoop()

    ...

    if player.rectangleCollisionDetection(target) then
        triggerWinState()
    endif

    ...

endprocedure
```

Sound effect when jumping:

This section of code will trigger a sound effect when the player jumps to make the game feel more responsive.

I have placed this section of code in the player method for jumping because this method handles everything that happens when the player jumps.

```
class player

    ...

    public procedure jump(..., jumpSound)

        if grounded and jumpKeyPressed then
            ...
            pygame.mixer.playSound(jumpSound)
        endif

    endprocedure

    ...

endclass
```

Sound effect when completing a level:

This section of code will trigger a sound effect when the player completes a level to make the game feel more responsive and rewarding.

I have placed this section of code in the game complete screen because this handles everything that happens when the player completes a level.

```
procedure levelComplete()

    ...

    levelCompleteSound = pygame.loadSound("levelcomplete.wav")
    pygame.mixer.playSound(levelCompleteSound)

    ...
```

```
endprocedure
```

Sound effect when dying:

This section of code will trigger a sound effect when the player fails a level to make the game feel more responsive.

I have placed this section of code in the game over screen because this handles everything that happens when the player fails a level.

```
procedure gameOver()

    ...

    levelFailSound = pygame.loadSound("levelfail.wav")
    pygame.mixer.playSound(levelFailSound)

    ...

endprocedure
```

Special offer:

This section of code will load a special offer to show the user to reward them for completing a level and promote Tekkerz. It will then display the image using pygame.

I have placed this section of code in the win state screen because this handles everything that happens when the player completes a level. This will also allow me to create local variables such as the variables used here such that they will not affect any other sections of code used to create the game.

```
procedure levelComplete()

    ...

    specialOfferImage = pygame.loadImage("specialoffer.jpg")
    pygame.displayImage(specialOfferImage)
```

```
    ...

endprocedure
```

Logo:

This section of code will load a logo to show the user to promote Tekkerz. It will then display the image using pygame.

I have placed this section of code in the main menu screen because this is the screen that the user will first see when they open the game, so the user will definitely see this promotion. It will also allow me to create local variables such as the variables used here such that they will not affect any other sections of code used to create the game.

```
procedure levelComplete()

    ...

    specialOfferImage = pygame.loadImage("specialoffer.jpg")
    pygame.displayImage(specialOfferImage)

    ...

endprocedure
```

Game over screen:

This section of code is just a procedure to display all the buttons and images on the game over screen, so there is no real logic to write pseudocode for. I will use a separate function for this piece of code and call it when the player fails the level because this will make the code more maintainable as this function can be called anywhere in the program and will always take the user to the game over screen.

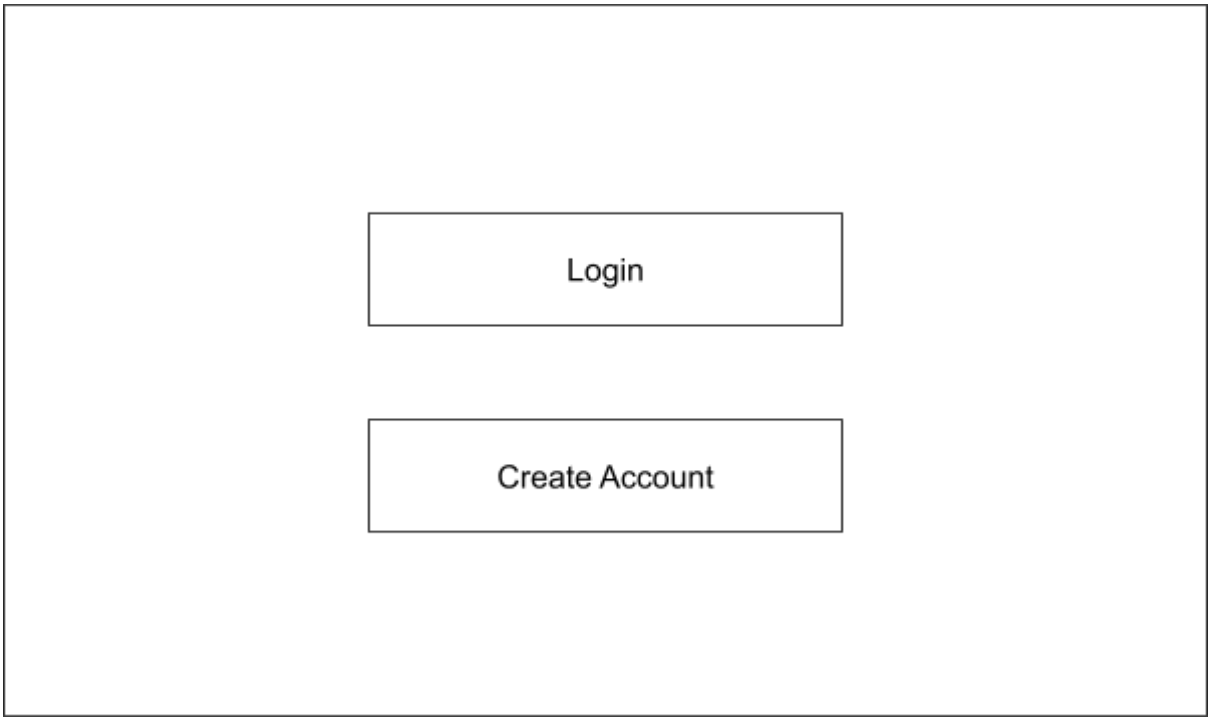
How this will form a complete solution:

These sub-problems with their solutions will form a complete solution because they cover all main parts of the desired solution. I have split the problem into “account management”, “in-game UI”, “levels”, “gameplay” and “other screens”. This is because these are the largest, most basic sections I can split the solution into. This subdivision ensures that I cover the

whole solution because I have an “other screens” section which will act as a miscellaneous section. Each of the sections of pseudocode represents a section of the solution with some logic that should be planned out because it contains some combination of selection, iteration, and recursion.

User Interface Design and Usability Features

Login or create account screen:

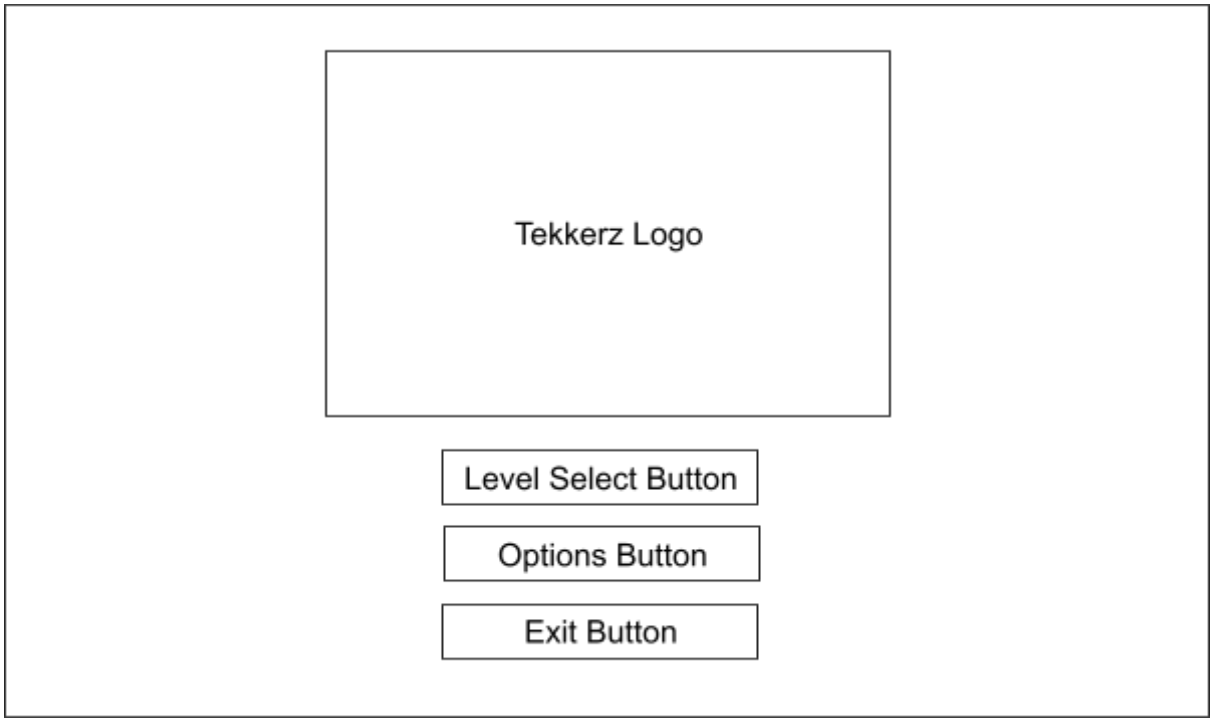


This will be a very simple screen because it should be intuitive and easy to use.

The Login button will be at the top because it will be the most used button on this screen.

The background will be white and the buttons will be white with a black outline and text to make them clearly readable. This will also make this screen really simple to use.

Main Menu:



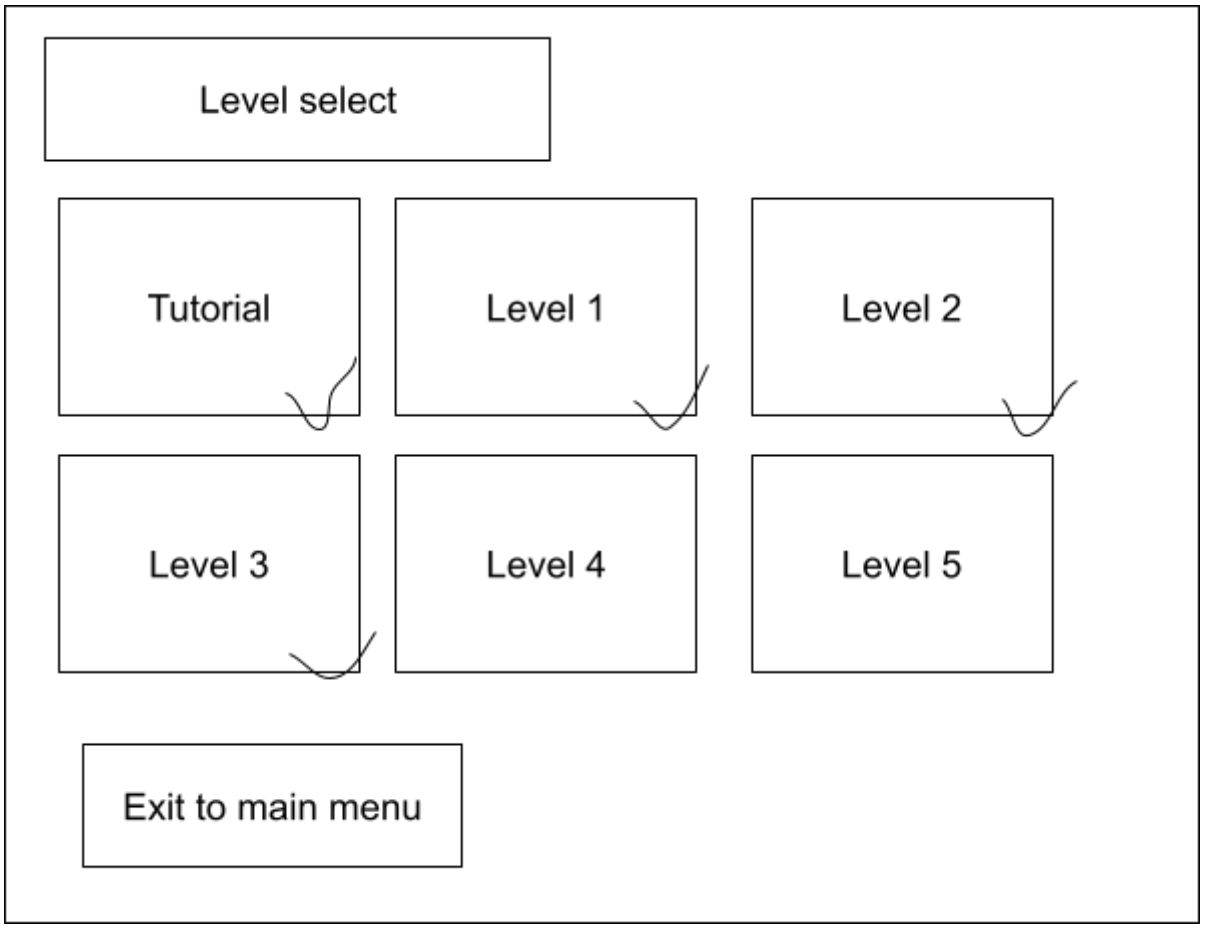
The Tekkerz Logo will be brightly coloured and the largest thing on the screen so it is as noticeable as possible and acts well as promotion for the restaurant

The Background will be black to create high contrast to make the logo and all the buttons stand out.

The level select button will be pale grey to stand out from the background and, but not be too harsh to look at. It will be the top button as it will be the button that is most commonly selected by the user.

The exit button will be red and at the bottom of the screen to discourage the user from quitting the game.

Level select screen:



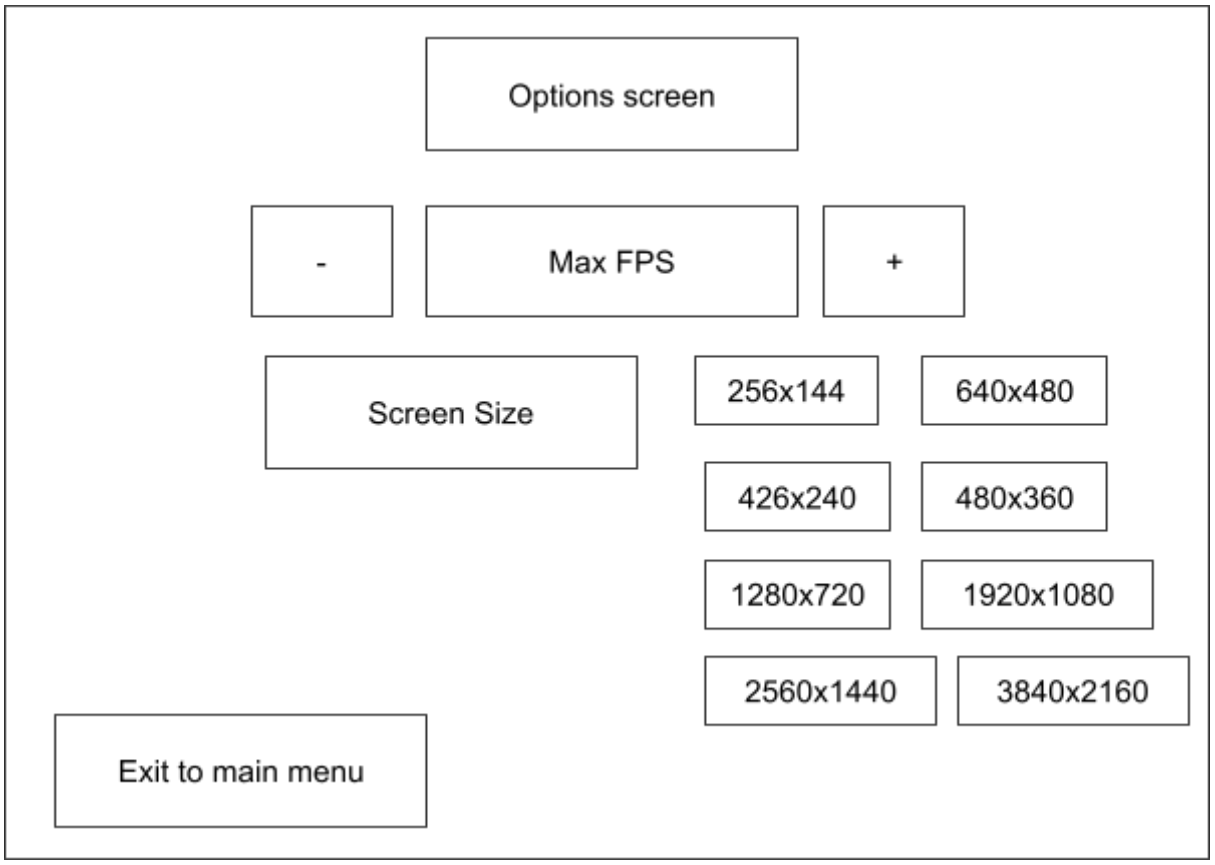
On this screen, there will be a large label in the top left to notify the user that they are on the level select screen.

Then there will be a button for each level. These will display the name of the level and whether the user has completed it or not. These should change colour when the user hovers over them to make it apparent that they can be clicked, otherwise they might just look like information boxed about the levels. When the user clicks on a level button, they will be taken to that level.

There will also be a button to exit back to the main menu. This will be red and at the bottom of the screen to discourage the user from quitting.

The background will be black, the buttons will be grey, and the text will be white, or any combination of colours that produces a high contrast so that the buttons will be easily readable.

Options screen:



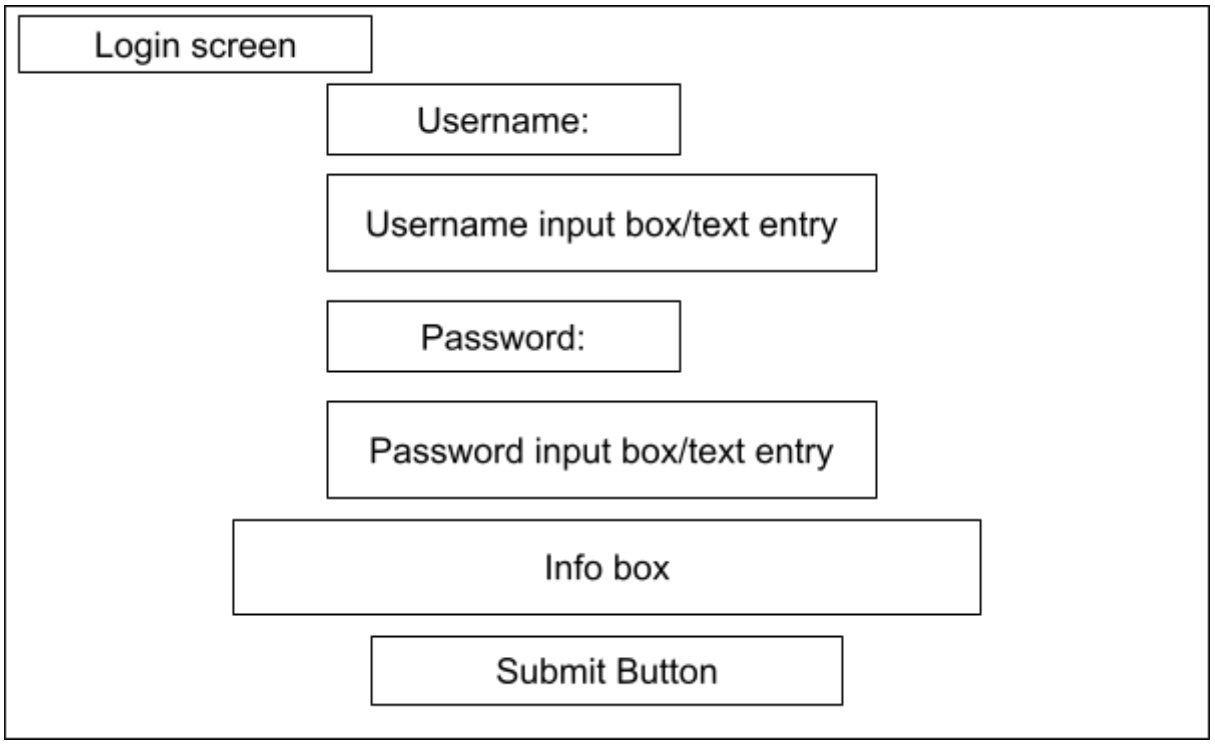
On this screen, there will be a large label in the top left to notify the user that they are on the options screen.

There will also be labels for options such as screen size and max FPS. And buttons to change these options, which will change colour to signify that they can be clicked.

The background will be black, the buttons will be grey, and the text will be white, or any combination of colours that produces a high contrast so that the buttons will be easily readable.

There will also be an “exit to main menu” button, which will be the same as the one on the level select screen.

Login screen:



This is the screen the player will be taken to if they click the login button on the “create account or login” screen.

This screen will contain a label/title at the top of the screen to alert the player as to what the data they enter will be used for.

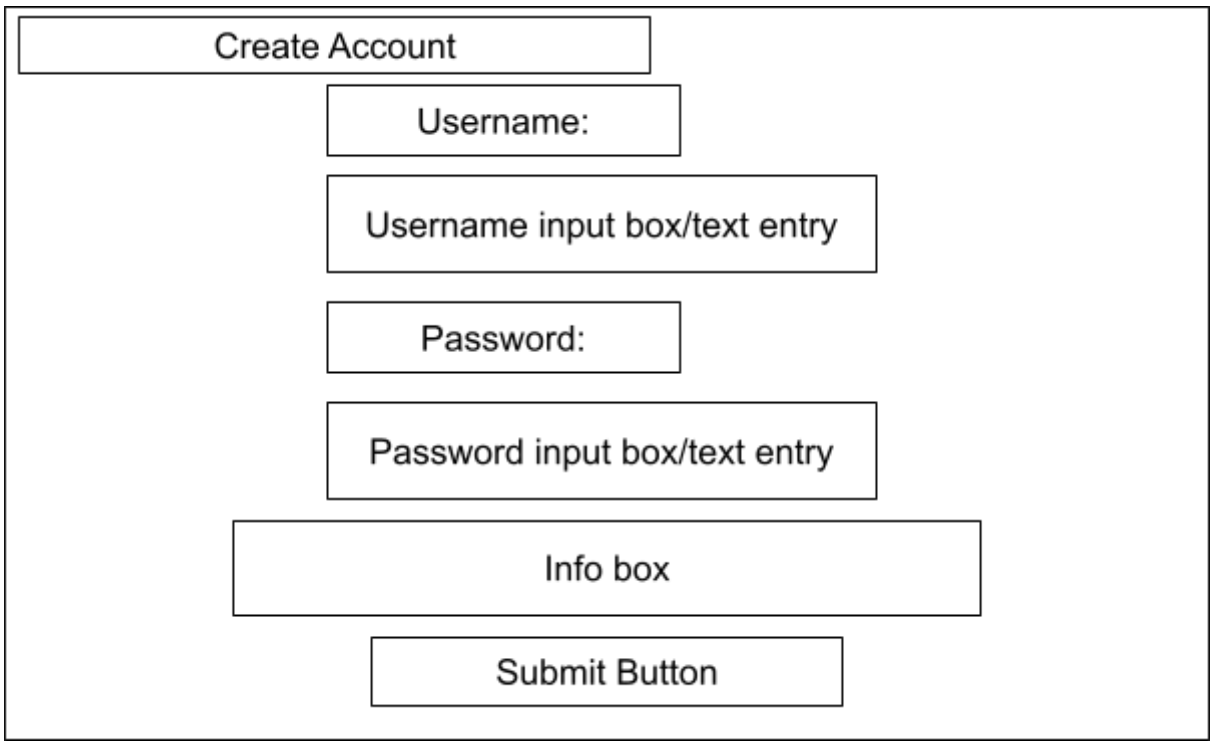
There will then be username and password entries with labels to tell the user what they should enter in each box. The text boxes should clearly show if they are selected with a marker to make sure the user does not input incorrect information.

There should also be an info box near the bottom of the screen which will display red text (to show that the message is a warning) to notify the user that the username or password they entered was wrong.

The submit button should be large and obviously clickable so that the user knows what to do.

This screen should be very simple, ideally black and white, to make the screen intuitive for the user and not over-complicated.

Create account screen:



This is the screen the player will be taken to if they click the create account on the “create account or login” screen.

This screen will contain a label/title at the top of the screen to alert the player as to what the data they enter will be used for.

There will then be username and password entries with labels to tell the user what they should enter in each box. The text boxes should clearly show if they are selected with a marker to make sure the user does not input incorrect information.

There should also be an info box near the bottom of the screen which will display red text (to show that the message is a warning) to notify the user that the username is already taken, or that the password is not secure enough.

The submit button should be large and obviously clickable so that the user knows what to do.

This screen should be very simple, ideally black and white, to make the screen intuitive for the user and not over-complicated.

Variables

Name	Data type	Where	Purpose/Justification
x	float	in the Vector class	The horizontal component of a position, velocity or acceleration.
y	float	in the Vector class	The vertical component of a position, velocity or acceleration.
pos	Vector	in the Rectangle class	The position of a rectangle.
size	Vector	in the Rectangle class	The size of the rectangle.
rect	Rectangle	in the Player class and the Button class	Data representing the position and size of any game object of screen element
vel	Vector	in the Player class	The velocity of the player (how much it moves per second).
acc	Vector	in the Player class	The acceleration of the player (how much the velocity will change each second).
sprite	Image	in Player class	The visuals of the player.
player	Player	in gameplay function	The player's state/data.
platforms	Rectangle[]	in gameplay function	The positions and sizes of all the platforms in a level.
kill_boxes	Rectangle[]	in gameplay function	The positions and sizes of all kill boxes (fail state triggers) in a level.
target	Rectangle	in gameplay function	The positions and size of the target (win state trigger).
text/label/title	pygame.Text	in Button class and in all menu functions including the main menu, the level select	The text on a button or as a label on the screen to inform the player what it does.

		screen and the options screen	
on_click	function	in Button class and in tkinter screens	To describe the behaviour of a button.
is_pressed	boolean	in Button class	Whether the button is currently being pressed by the player's mouse.
level_buttons	Button[]	in the level select function	The buttons used to open the levels.
level_select_button	Button	in the main menu function	The button the user will click on to get to the level selection screen.
options_button	Button	in the main menu function	The button the user will click on to get to the options screen
max_fps_plus_button	Button	in the options screen function	The button the user will click on to increase the max FPS
max_fps_minus_button	Button	in the options screen function	The button the user will click on to decrease the max FPS
resolution_buttons	Button[]	in the options screen function	The buttons the user will click on to change the screen's resolution.
username	string	in the User class	The display name and unique user identifier for a user.
password	string	in the User class	The string the user must know to access their account.
progression	dictionary	in the User class	Data about where the user has got to in certain levels and whether they have completed the level. It will have key value pairs of level_id: LevelProgression.
completed	boolean	in the LevelProgression class	Whether a given user has completed a level.
pos	Vector	in the LevelProgression	Where the player is in a level.

		class	
vel	Vector	in the LevelProgression class	The velocity the user has in a Level (so that the user starts exactly where they left off).
dt	float	in all pygame screens (mostly used in the gameplay function)	The time elapsed between the last frame and the current frame (delta time).
WIDTH	constant int	in all pygame screens	The width of the screen in pixels.
HEIGHT	constant int	in all pygame screens	The height of the screen in pixels.
mouse_pos	Vector	in the gameplay loop	the position of the mouse at the start of a given frame.
keys_pressed	dictionary	in the gameplay loop	A dictionary that maps keys to booleans indicating whether they are pressed.

Validation

Purpose	Validation Type	Justification	Name of Variable
Checks that the length of the password is at least 5 characters long.	Length check	This will make the password adequately secure for a game (not critical).	password
Checks that the username contains no spaces.	Format check	This will make sure the username can be stored correctly.	username
Checks that the password contains no spaces.	Format Check	Passwords with spaces may not be properly stored or represented.	password
Checks that a password is entered before the account can be created.	Presence Check	The user must have a password to protect their account. (This may be covered by the	password

		length check.)	
Checks that the user has entered a username before the account can be created	Presence Check	The user must have a username to uniquely identify them.	username
Checks that the mouse is within the boundaries of the button when clicked.	n/a	For the button to be clicked, the mouse must be within the boundaries of the button, or else nothing will happen.	mouse_pos
Checks that a specific key has been pressed e.g. a space for jumping and WASD for movement.	n/a	For an action in the game to be taken, a specific key must be pressed, or else nothing will happen.	keys_pressed

Testing

Testing and data to be used in the post-development phase:

Test Number	Description	Test Data (if applicable)	Justification
1	Does the password checker reject and accept and reject the right passwords?	“abc” (erroneous), “hello” (boundary), “password123” (valid)	Passwords must be rejected if their length is less than 5 (or some other number) to ensure security.
2	Can the text entries be edited, and can the changes be checked.	Typing in the text entry while it is selected. (valid) Typing while the text entry is not selected (boundary)	For the user to enter their details, I must be able to check the text entries’ contents when the submit button is pressed.
3	Does clicking the submit button on the create account screen create and save a user to a file in secondary storage.	Clicking the button with valid details. (valid) Clicking the button with invalid details. (boundary)	A new user should be saved to a file in secondary storage when the button is clicked so that the user can login later, but only when the

			details in the text entries are valid.
4	Does the program load the users from the file in secondary storage into main memory?	n/a	The program needs access to the users' data to check entered login details against.
5	Is the correct user selected when the user enters their login details.	Username and password correct. (valid) Username correct but password incorrect. (boundary) Username not in the list of users and any password. (erroneous)	The entered username in the login screen must be used to select a user from the list of users with the same username (if there is one). And then check that the passwords match. This will ensure that users can only get access to their own account (the one they know the username and password for).
6	Do the buttons correctly call their on click function when clicked, and do nothing when not clicked.	Clicking on the button. (valid) Holding down the button, moving out of the button and then releasing. (boundary) Clicking outside the button, moving inside the button and then releasing. (boundary)	The buttons must only activate when the user both clicks inside the button and releases inside the button. This is because this is the most common way buttons work in other programs, so it will be intuitive.
7	Are the platforms displayed correctly on the screen? I will visualise where I expect the platform to appear, then test if that is where the platform is actually drawn.	n/a	The platforms must conform to the coordinate system of the level, because the player has to interact with them (such as during collisions).
8	Do the levels get loaded correctly from a JSON file in secondary storage into a list in main memory.	A properly formatted JSON file with all the correct fields in each of the levels. (valid)	Because all objects in the level must be where specified in the JSON file. If there are any

		A properly formatted JSON file without all the required fields. (boundary) An improperly formatted JSON file. (erroneous)	missing fields or invalid JSON in the JSON file, the level loading should stop, and an error should be thrown. This is to ensure that no error-prone or incomplete levels are loaded and played by the user.
9	Does the user jump correctly (only when on the ground and once when in the air, if a double jump is implemented)?	Pressing the jump input (spacebar) when on the ground. (valid) Pressing the jump input when in the air (after the double jump if applicable) (boundary)	The player must not be able to jump when the game should not allow it, as the levels would not be designed for this movement, and the player may be able to take unintended shortcuts to finish the level, making the game less engaging.
10	Does the player move horizontally correctly?	Pressing left input and right input and both at the same time.	The player must move in a predictable manner, so the user can get used to the player's movement more easily, making the game more engaging. (e.g. Not moving when the left and right input are both pressed.)
11	Are collisions detected and responded to correctly? i.e. Does the player get blocked by platforms, get killed by kill boxes when entering, win when colliding with the target, e.t.c.	Moving at a normal pace towards a platform. (valid) Moving really fast towards the platform. (boundary, to test no-clip resistance)	I must test that all interactions involving collisions work, to give the game as many working features as possible, to make it more engaging.
12	Is the player’s movement frame-rate independent. (This includes both constant velocity and when the player is accelerating.)	Setting the game to 60 fps. (valid) Setting the game to 10 fps. (boundary)	The game should work on as wide a range of hardware as possible, so that the game is

			accessible to as many people as possible.
13	Is an appropriate sound played when the player jumps?	n/a	The game should give appropriate feedback to the player when they take an action to make the game more responsive.
14	Is an appropriate sound played when the player dies?	n/a	The game should give appropriate feedback to the player when something happens to the player to make the game more responsive.
15	Is an appropriate sound played when the player completes a level?	n/a	The game should give appropriate feedback to the player when they take an action to make the game more responsive.
16	Is some promotional material shown (a special offer when the player completes a level and the Tekkerz logo when on the main menu screen)?	n/a	The game is made to promote the Tekkerz restaurant change, so must show promotional material effectively.
17	Is the game over screen shown when the user dies in a level?	n/a	The game should give appropriate feedback to the player when something happens to the player to make the game more responsive.
18	Does all movement work correctly?	Pressing the wasd and spacebar keys.	Movement must be fully functional for the user to complete the levels.
19	Does the user get a special offer after completing a level?	Hitting the target and completing the level.	The stakeholder specified that the game must have sufficient

			promotional material.
20	Is all user progression saved correctly?	Making progress, saving and then exiting.	The user must be able to leave the game and return to where they left off to make the game more engaging.
21	Can the user double jump?	Pressing the jump key while in the air.	This is an extra ability that will keep the user engaged.

Iterative testing:

Test number	Test Description	Test data (if applicable)	Justification
1.1	Creating some vectors and then adding them.	numbers to create the vectors: 2, 4, 3, 5 (valid)	All vector operations must be correct for physics calculations to work.
2.1	Create a rectangle and print out the data it contains.	numbers to create the rectangle, 0, 0, 20, 10 (valid)	Data must be stored in the rectangle correctly to properly represent game objects so they can be drawn and interacted with.
3.1	Create a list containing some data, print some information about it. Then save it to secondary storage and load it from secondary storage.	the list used to store data, [1, 5, 2, 5, 3] (valid)	Data must remain consistent when being stored to and loaded from secondary storage.
4.1	Does the blank pygame screen code work?	n/a	I need to create a blank pygame screen to build on top of to make the game.
5.1	Does the player appear on the screen and move?	The player's velocity (10, 10). Valid	I must be able to give the player some velocity and have it move on the screen as this will be the basis for the player's movement.
6.1	Does the player collide correctly with the platform?	n/a	The player needs to collide with platforms to be able to stand on them and be able to hit them, so they can act as an obstacle.

7.1	Does the player accelerate downwards properly when being accelerated by gravity.	The strength of gravity (some arbitrary number e.g. 100). Valid	The player needs to accelerate downwards to make jumping realistic. This movement should also stay frame rate independent.
8.1	Does the discrete collision process still work when used in conjunction with a low framerate and thin platform?	The low framerate e.g. 10 (boundary)	If the collision doesn't work, the game will run differently on low-end machines that can't run the game at a high frame rate. This is not ideal.
8.2	Does the collision work horizontally and when in a corner (moving into a wall)?	The new platform at an arbitrary position (valid).	The platforms must act as barriers to get over as well as traditional platforms to provide an obstacle for the player.
9.1	Do the player's horizontal movement inputs make the player move correctly?	The keys the player pressed (A and D) (valid).	The player must be able to move horizontally to move around and complete the level.
9.2	Do the player's horizontal movement inputs make the player move correctly?	The wrong keys e.g. J and L (erroneous)	The player must not move if the wrong keys are pressed.
10.1	Can the player jump to clear an obstacle?	The user pressing space. (valid)	The player must be able to jump to overcome obstacles.
11.1	Does the player gradually slow down while moving due to the air resistance and fraction added?	The player's movement (applied by the user). (valid)	The player must not slide forever after picking up speed. This will make the game easier for beginners and more intuitive.
12.1	Can the screen zoom in and out?	The amount of zoom applied e.g. 40 (valid)	The screen must be able to zoom in and out to accommodate for different resolutions.
12.2	Does the zoom take into account the screen resolution?	The screen resolution e.g. 300x200 (valid)	The user must be able to see the same amount of the level on all different resolutions.
13.1	Does the sample text show up on the screen?	The sample text, colour and font ("A", BLACK, Comic Sans MS) (Valid)	The menu screens must display text to show the user what screen they are on and how to interact with the manu to achieve their desired outcome.
13.2	Is the prototype menu screen shown when a	The key pressed, "A". (valid)	The program must be able to switch screens by calling the

	key is pressed?		function corresponding to a menu screen within the parent screen's function.
14.1	Does the button display properly?	The rectangle and text of the button "A" (valid)	The buttons must be visible for the user to use it.
14.2	Does clicking the button take the user to the associated screen?	n/a	The user must be able to click buttons to navigate the menu screens.
14.3	Does the main menu load and display the Tekkerz logo	the image file name, "logosmall.png" (valid)	To promote the game Tekkerz, the logo should be shown on the man menu (the user will see this whenever the game loads).
15.1	Do the buttons display properly and in the right positions?	The positions and dimensions of the buttons along with their text, ("options", WIDTH / 2, 5 * HEIGHT / 10, 200, 50) (valid)	The buttons must be displayed properly for the user to use them easily.
16.1	Creating an example user and serialising it to save in secondary storage. Does an example user object (used to group data)?	The data in the user (username="Jonathan", password="asdfgh", progression=None) (valid)	The program must correctly save user data so they can quit and login later.
17.1	Does the login code correctly show a screen and validate the user's input?	The input Jonathan and "asdfgh" (valid)	The user must be able to login.
18.1	Can the login code count how many attempts have been taken.	n/a	The number of attempts must be recorded to prevent a brute force attack.
18.2	Does the login screen message update based on the number of attempts remaining?	the number of attempts remaining (3, 2 or 1) (valid)	The user should be told how many attempts they have remaining before they are kicked out of the screen.
19.1	Does the account options screen show?	n/a	I need an account options screen so the user can select whether to login or create an account.

19.2	Does clicking a button on the account options screen run the corresponding function and close the screen properly?	n/a	The account options screen should close after selecting an option to avoid screen clutter.
20.1	Can the create account screen successfully take an input and print out the input data?	The data entered into the account creation text entries. (username=”a_name”, password=”a_password”) (valid)	I need an account options screen so the user can select whether to login or create an account.
20.2	Does the create account screen properly reject an input request to create an account with an existing username?	The data entered into the account creation text entries. (username=”jonathan”, password=<anything>) (valid)	The username is a unique identifier for an account, so duplicate usernames should not be allowed.
20.3	Does the account creation screen accept a password of 5 characters or more?	An adequate password “amskdlaksdjlaksd” (valid)	Making sure the account creation still works after adding the password security checker.
21.1	Is the level data loaded correctly?	Data in the levels.json file (valid)	The level data must be loaded from secondary storage where it is stored to main memory where it will be processed and used.
22.1	Is the level data that has been loaded from secondary storage processed and displayed properly by the gameplay function.	Data in the level object. (valid)	The level data must be properly processed so that all the levels can be used by the game and provide adequate content for the user.
23.1	Does the level select button take the user to the level select screen?	n/a	The user must be able to navigate to the level select screen to be able to select a level?
23.2	Does clicking on a level button take the user to the associated level?	n/a	The level select buttons must take the user to a level so that the user can access the levels.
24.1	Does the target show up in a level?	the position of the target (10, -1)	The user must be able to see the target to know how to

		(valid)	complete the level. This also shows that the game code can properly process the target.
24.2	Are collisions detected properly between the player and the target?	n/a	The collision detection between the player and the target must work all the time for the user to be able to complete the level.
25.1	Does the level 2 button appear and load level 2 when clicked?	n/a	I must have multiple levels to keep the user engaged.
26.1	Does the level ordering system using the levels order JSON file work?	The data in the levels order JSON file [“0”, “1”, “2”] (valid)	The levels must be ordered correctly so there can be some sensible difficulty progression.
27.1	Do all the necessary buttons show up when running the pause screen?	n/a	All the buttons on the pause screen must be shown so that the user can navigate and control the program.
28.1	Is the level progression loaded correctly when entering a level?	The level data (user.progression[“1”]) (valid)	The progression needs to be loaded into the level correctly to know where the player should be and other elements of progression.
28.2	Is the level progression updated when the user hits the target?	n/a	The progression needs to be updated when the user completes the level so that the user can be told which levels they have completed in the future.
28.3	Does a prompt to save to secondary storage pop up and function properly when the user tries to exit the game?	n/a	The user may exit the game without saving by accident and lose their progress.
28.4	Does a save button appear and function correctly on the main menu?	n/a	The user may want to save manually without the game closing.
29.1	Does the game play a sound effect when the player jumps?	The name of the sound file to be used, “jumpsound.wav” (valid).	The game should play a sound when the player jumps to make the game more responsive.
30.1	Can the game detect	The level bounds	The game needs to detect

	when the player is out of bounds?	e.g. (x:0, y:0, hw: 100, hh: 100) (valid)	when the player goes out of bounds so it can be killed.
30.2	Does the game restart properly when clicking retry on the game over screen?	n/a	The player should be able to have another go at the level after failing to keep them engaged.

30.3	Does hitting a kill box kill the player?	The positions of the kill-boxes e.g.
		[
		{
		"x": 2,
		"y": -1,
		"hw": 1,
		"hh": 0.2
		}
		] (valid)
			This will provide a fail condition so that the player can lose the level, this will give the levels an element of challenge, so they are not too easy and will keep the user engaged.

31.1	Is a sound played and a special offer image shown when the user completes a level?	The name of the image and sound files “assets/specialoffer.png” and "assets/levelcompletesound.wav" (valid)	The player should be rewarded when completing a level.
32.1	Is the player’s position a velocity saved when exiting a level?	The player’s position and velocity e.g. pos = 10, 10 velocity = 5, 20 (valid)	The player should be able to resume where they left off.

Development

Vector

I need a 2D vector class to represent the positions and velocities of the player.

```
class Vector:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    # methods for vector arithmetic
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y)

    def __mul__(self, k):
        return Vector(self.x * k, self.y * k)

    def __truediv__(self, k):
        return self * (1 / k)

    def __str__(self):
        return f'({self.x}, {self.y})'
```

This allows the use of arithmetic operators on the vector

Test:

```
a = Vector(2, 4)
b = Vector(3, 5)
print(a + b)
```

Output:

(5, 9)

A method to get the angle of a vector to help with physics calculations.

```
from math import atan
```

```
    def angle(self):
        return atan(self.y / self.x)
```

Test:

```
a = Vector(2, 4)
b = Vector(-2, -4)
```

```
print(a.angle(), b.angle())
```

Output:
The two angles are the same, and they should not be. The numbers are low because the math.atan2 function returns angles in radians.
1.1071487177940904 1.1071487177940904 (Logic Error)

This doesn't work because

```
self.y / self.x
```

Is the same for both a and b

This can be fixed using the math.atan2 function
Fig 1.1

```
from math import atan2
```

```
    def angle(self):
        return atan2(self.y, self.x)
```

Test:

```
a = Vector(2, 4)
b = Vector(-2, -4)
print(a.angle(), b.angle())
```

Output:
1.1071487177940904 -2.0344439357957027

And some extra methods

```
    def __iter__(self):
        return iter([self.x, self.y])

    def unwrap(self):
        return self.x, self.y

    def copy(self):
        return Vector(self.x, self.y)

    def angle(self):
        return atan2(self.y, self.x)

    @staticmethod
    def dot(v1, v2):
        return v1.x * v2.x + v1.y * v2.y

    def rotate90(self):
        return Vector(-self.y, self.x)

    def reflect_x(self):
```

```
        return Vector(-self.x, self.y)

    def reflect_y(self):
        return Vector(self.x, -self.y)

    def flip(self):
        return Vector(-self.x, -self.y)
```

This method

```
    @staticmethod
    def dot(v1, v2):
        return v1.x * v2.x + v1.y * v2.y
```

Calculates the dot product of two vectors which can help with collision detection.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
1.1	Creating some vectors and then adding them.	numbers to create the vectors: 2, 4, 3, 5 (valid)	All vector operations must be correct for physics calculations to work.	As Expected
1.2	Creating a vector and calculating its angle to the horizontal	Numbers used to create the vectors (valid)	All vector operations must be correct for physics calculations to work.	Failed and fixed using the math.atan2 method (**Fig 1.1**)

Rectangle

A Rectangle class can represent the player's hitbox, a platform or the entire level.

```
from game.vector import Vector



class Rectangle:

    def __init__(self, x, y, hw, hh):
        self.pos = Vector(x, y)
        self.size = Vector(hw, hh)
    # a method to print the rectangle as a string for debuggong
    def __str__(self):
        return f'Rectangle: (pos: {self.pos} dim: {self.size})'
```

pos is a vector representing the position of the centre of the rectangle.
size is a vector from the centre of the rectangle to the bottom right hand corner.

Test:

```
r = Rectangle(0, 0, 20, 10)
print(str(r))
```

Output:
Rectangle: (pos: (0, 0) dim: (20, 10))

This is a method used to calculate the position of a corner of a rectangle.
I will use iteration here because I can repeatedly switch corners until I get to the right one.

```
    def corner_pos(self, n):
        corner_vector = self.size.copy()
        for i in range(n):
            corner_vector = corner_vector.rotate90()
        return self.pos + corner_vector
```

Test:

```
r = Rectangle(0, 0, 20, 10)
print(r.corner_pos(1))
```

Output:
(-10, 20) (Logic Error)
Expected:
(-20, 10)

This doesn't work because the position of the bottom left corner is obtained by reflecting the corner_vector, not by rotating it

Instead, I use the following which are separate methods for each corner.
Fig 2.1

```
    def bottom_right_pos(self):
```

```
        return self.pos + self.size

    def bottom_left_pos(self):
        return self.pos + self.size.reflect_x()

    def top_left_pos(self):
        return self.pos + self.size.flip()

    def top_right_pos(self):
        return self.pos + self.size.reflect_y()
```

Test:

```
r = Rectangle(0, 0, 20, 10)
print(r.bottom_left_pos())
```

Output:

(-20, 10)

Here are the other methods

```
    def unwrap(self):
        return self.pos.unwrap(), self.size.unwrap()

    def corner_rect_tuple(self):
        return (self.left_pos(), self.top_pos(), self.size.x * 2,
self.size.y * 2)

    def combine_rectangle(self, other):
        return Rectangle(self.pos.x, self.pos.y, self.size.x +
other.size.x, self.size.y + other.size.y)

    def contains_point(self, p):
        return not (p.x < self.pos.x - self.size.x or p.y < self.pos.y
- self.size.y or p.x > self.pos.x + self.size.x or p.y > self.pos.y +
self.size.y)

    def right_pos(self):
        return self.pos.x + self.size.x

    def bottom_pos(self):
        return self.pos.y + self.size.y

    def left_pos(self):
        return self.pos.x - self.size.x

    def top_pos(self):
```

```
        return self.pos.y - self.size.y
```

Test number	Test Description	Test data (if applicable)	Justification	Outcome
2.1	Create a rectangle and print out the data it contains.	numbers to create the rectangle, 0, 0, 20, 10 (valid)	Data must be stored in the rectangle correctly to properly represent game objects so they can be drawn and interacted with.	As Expected
2.2	Creating a rectangle and getting the position of one of its corners using the corner_pos method	the number of the corner e.g. 1 (valid)	I must be able to obtain the position of the rectangle's corners to process collisions.	Failed and fixed (**Fig 2.1**) by separating one method out into separate methods.

Serialisation

These are some helper functions to load and save a python object to a binary file using serialisation.
This will be used to save user data and progression in levels.

```
import pickle

# load and return an object in the form of a pickle dump at this file path
def load_obj(file_path):
    with open(file_path, r"rb") as f:
        users = pickle.load(f)
    return users
# save an object into a file as a pickle dump
def save_obj(obj, file_path):
    with open(file_path, r"wb") as f:
        pickle.dump(obj, f)
```

Pickle is a serialisation library.

Test:

```
# create some data
l1 = [1, 5, 2, 5, 3]
print(l1, type(l1), id(l1))
# save it to a file
save_obj(l1, r"./afile.p")
# and load it back into main memory
l2 = load_obj(r"./afile.p")
print(l2, type(l2), id(l2))
```

Output:
[1, 5, 2, 5, 3] <class 'list'> 2181057230336
[1, 5, 2, 5, 3] <class 'list'> 2181057307264

l1 and l2 do not maintain the same reference (they are different objects in memory even though they have the same values).

Test number	Test Description	Test data (if applicable)	Justification	Outcome
3.1	Create a list containing some data, print some information about it. Then save it to secondary storage and load it from secondary storage.	the list used to store data, [1, 5, 2, 5, 3] (valid)	Data must remain consistent when being stored to and loaded from secondary storage.	As Expected

Basic player movement around a level - Prototype 1, A blank pygame screen

Using some boilerplate found online (https://iqcode.com/code/python/pygame-boilerplate), I made a simple pygame screen. I will use iteration here to run the game loop because the code in the game loop must be run once per frame until the game stops. And I will use it when checking the events because there are an arbitrary amount of events each frame that need to be processed in turn.

```
WIDTH = 1000
HEIGHT = 600

FPS = 60

BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
clock = pygame.time.Clock()

# the game loop, this runs continuously as the game plays
running = True
while running:
    # delta time, the time between frames to
    dt = clock.tick(FPS) * 0.001
    # the event loop used
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()

    screen.fill(BLACK)

    pygame.display.flip()

pygame.quit()
```

Output:

Test number	Test Description	Test data (if applicable)	Justification	Outcome
4.1	Does the blank pygame screen code work?	n/a	I need to create a blank pygame screen to build on top of to make the game.	As Expected

Basic player movement around a level - Prototype 2, A moving shape

A player class (for now this will just be a rectangle that moves across the screen)

```
class Player:
    def __init__(self, x, y, hw, hh, vel_x=0, vel_y=0):
        self.rect = Rectangle(x, y, hw, hh)
        # the velocity of the player (how it moves each frame)
        self.vel = Vector(vel_x, vel_y)

    def update_position(self):
        self.rect.pos = self.rect.pos + self.vel
```

With this added to the game

```
player = Player(100, 100, 20, 20, vel_x=10, vel_y=10)
```

And this added to the game loop

```
player.update_position()
pygame.draw.rect(screen, WHITE, player.rect.corner_rect_tuple())
```

Output:
(The player moves across the screen diagonally down and to the right.)

Test:

```
FPS = 30
```

and

```
FPS = 10
```

Output:
The white square moves slower. (Logic Error)

Fig 5.1

This movement is dependent on the game's framerate ie the player will move slower if FPS = 30 than if FPS = 60 because the player's velocity will be applied fewer times per second, to make it framerate independent, I add

```
self.dis = Vector(0, 0)
```

to the player class's initialisation method and this method to the player class, along with changing the update_position method

```
def update_displacement(self, dt):
    self.dis = self.vel * dt

def update_position(self):
    self.rect.pos = self.rect.pos + self.dis
```

And the game loop is changed to this

```
player.update_displacement(dt)
player.update_position()
pygame.draw.rect(screen, WHITE, player.rect.corner_rect_tuple())
```

This means the distance the player moves in a frame is proportional to dt (the time between frames), so the player's speed is independent of frame rate.

Test:

```
FPS = 30
```

Output:

The square moves at the right speed.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
5.1	Does the player appear on the screen and move?	The player's velocity (10, 10). Valid	I must be able to give the player some velocity and have it move on the screen as this will be the basis for the player's movement.	As Expected
5.2	Is the speed of the player's movement frame rate independent?	FPS 30 (valid), FPS 10 (boundary)	The player must move at the same speed on different computers that can run the game at varying frame rates.	Failed and fixed by using a displacement attribute and delta time (**Fig 5.1**)

Basic player movement around a level - Prototype 3, Discrete collision

http://jeffreythompson.org/collision-detection/rect-rect.php
Another rectangle is needed for the player to collide with (this will be a platform).
This is added to the game code

```
platform = Rectangle(150, 200, 200, 10)
```

And the game loop is changed to this

```
player.update_displacement(dt)
player.update_position()
# draw the player and platform
pygame.draw.rect(screen, WHITE, player.rect.corner_rect_tuple())
pygame.draw.rect(screen, BLUE, platform.corner_rect_tuple())
```

Output:
(The player moves through the platform) (Logic Error)

Fig 6.1
Now, collision logic is needed in the player script

```
def collide_with_rectangle(self, rect):
```

I create a rectangle which combines the two rectangles to simplify the problem into collision between a point and a rectangle

```
"""
two rectangles colliding can be simplified to a rectangle and point
colliding
"""
# combined rectangle
r = rect.combine_rectangle(self.rect)
# point
p = self.rect.pos
```

Then I get the positions of the sides of the rectangle

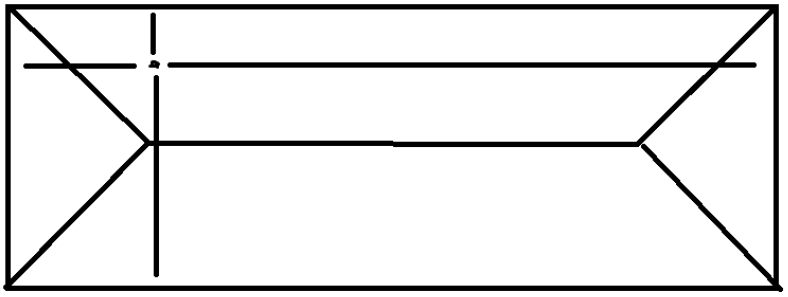
```
# positions of the sides of the rectangle
```

```
right = r.right_pos()
bottom = r.bottom_pos()
left = r.left_pos()
top = r.top_pos()
```

There can't be a collision if the point is outside the rectangle so the method returns early

```
# early return if point is outside the rectangle
if (
    (p.x > right)
    or (p.y > bottom)
    or (p.x < left)
    or (p.y < top)
):
    return
```

If the program has got to here, the rectangles must be colliding, then it calculates the distances of the point to the different sides of the rectangle to see which is closest.
This diagram shows the regions in which the point must be to collide with each side and how this distance comparison can facilitate this decision.



```
right_dist = right - p.x
bottom_dist = bottom - p.y
left_dist = p.x - left
top_dist = p.y - top

min_dist = min(right_dist, bottom_dist, left_dist, top_dist)
```

The player will collide with the closest wall

```
if min_dist == right_dist:
    # collision with right wall of rectangle
    pass
elif min_dist == bottom_dist:
    # collision with bottom wall of rectangle
    pass
elif min_dist == left_dist:
    # collision with left wall of rectangle
    pass
else:
```

```
    # collision with top wall of rectangle
    pass
```

To collide with a wall, the player's position must be set to be next to that wall and the component of its velocity which was sending it into that wall must be set to 0.
So this is the code to collide with the right wall

```
p.x = right
self.vel.x = 0
```

p can be used to change the position of the player's rectangle as it references Player.rect here

```
p = self.rect.pos
```

With all the responses to collision added, the final section looks like this

```
if min_dist == right_dist:
    # collision with right wall of rectangle
    p.x = right
    self.vel.x = 0
elif min_dist == bottom_dist:
    # collision with bottom wall of rectangle
    p.y = bottom
    self.vel.y = 0
elif min_dist == left_dist:
    # collision with left wall of rectangle
    p.x = left
    self.vel.x = 0
else:
    # collision with top wall of rectangle
    p.y = 0
    self.vel.y = 0
```

The game loop is now changed to this

```
player.update_displacement(dt)
player.update_position()
player.collide_with_rectangle(platform)
pygame.draw.rect(screen, WHITE, player.rect.corner_rect_tuple())
pygame.draw.rect(screen, BLUE, platform.corner_rect_tuple())
```

Output:
The player now teleports to the top of the screen when it collides with the top of the rectangle (Logic Error)

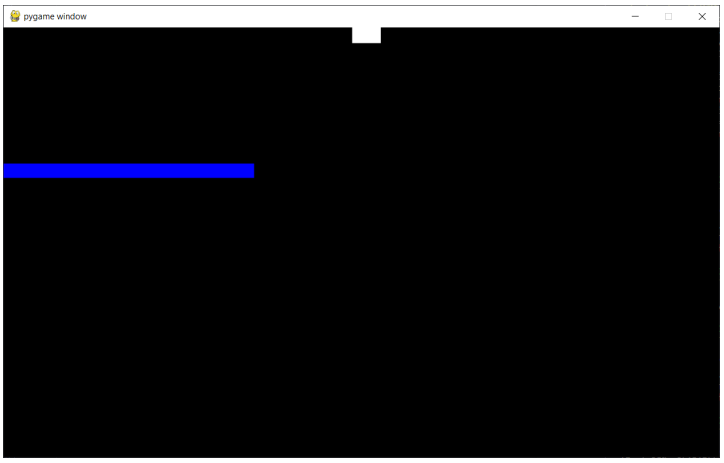


Fig 6.2

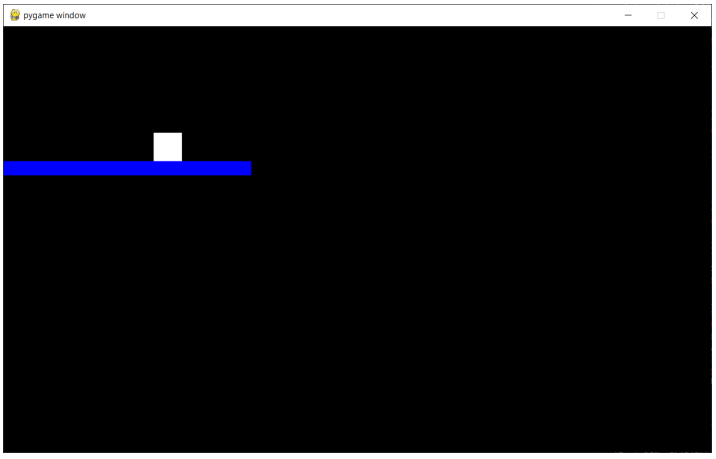
This is because this

```
    p.y = 0
    self.vel.y = 0
```

Should be

```
    p.y = top
    self.vel.y = 0
```

Output:



It now collides as intended.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
6.1	Does the player collide correctly with the platform?	n/a	The player needs to collide with platforms to be able to stand on them and be able to hit them, so they can act as an obstacle.	Failed and fixed by adding collision logic (**Fig 6.1**)
6.2	Does the player collide correctly with the platform after adding collision logic?	n/a	The player needs to collide with platforms to be able to stand on them and be able to hit them, so they can act as an obstacle.	Failed and fixed by changing the position correction (**Fig 6.2**)

Basic player movement around a level - Prototype 4, acceleration and gravity

To test the collision detection further, I will add gravity so the player falls onto the platform. For this, the player must experience acceleration, so its initialisation method is changed to

```
def __init__(self, x, y, hw, hh, vel_x=0, vel_y=0):
    self.rect = Rectangle(x, y, hw, hh)
    self.vel = Vector(vel_x, vel_y)
    self.acc = Vector(0, 0)
    self.dis = Vector(0, 0)
```

The update_displacement method is changed to

```
def update_displacement_and_velocity(self, dt):
        # s = ut when acceleration = 0
        self.dis = self.vel * dt
        # v = u + at
        self.vel = self.vel + self.acc * dt
```

These methods are added to the player class to enable acceleration and gravity

```
def apply_acc(self, acc):
    self.acc = self.acc + acc

def reset_acc(self):
    self.acc = Vector(0, 0)

def gravity(self, g):
    self.apply_acc(Vector(0, g))
```

And the game loop is changed to

```
player.reset_acc()
#acceleration
player.gravity(100)
# update displacement and velocity
player.update_displacement_and_velocity(dt)
player.reset_grounded()
# update position
player.update_position()
# collisions
player.collide_with_rectangle(platform)
```

Test:
Running this code with the gravity specified in the code.
Output:
The player's movement is no longer framerate independent because acceleration is not 0. (Logic Error)

Fig 7.1

So now the update_displacement_and_velocity method must be changed to this

```
def update_displacement_and_velocity(self, dt):
    # s = ut + 0.5at2
    self.dis = self.vel * dt + self.acc * (0.5 * dt * dt)
    # v = u + at
    self.vel = self.vel + self.acc * dt
```

Output:

This makes the player's movement frame rate independent again.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
7.1	Does the player accelerate downwards properly when being accelerated by gravity.	The strength of gravity (some arbitrary number e.g. 100). Valid	The player needs to accelerate downwards to make jumping realistic. This movement should also stay frame rate independent.	Failed and fixed by changing the equations of motion used. (**Fig 7.1**)

Basic player movement around a level - Prototype 4, continuous collisions

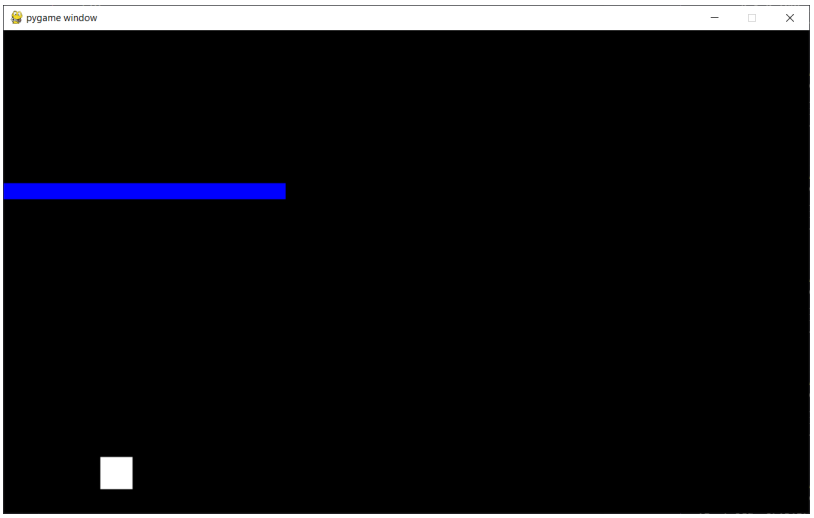
Test:
Running the game at a low frame rate with a thin platform.

```
FPS = 10
```

```
platform = Rectangle(150, 200, 200, 10)
```

Output:

Due to the way the collisions were processed previously, if the frame rate is too low, the player is moving too fast or the platform or player are too thin (in the direction they are travelling), the player will not collide with the platform correctly. (Logic Error)



As demonstrated in the diagram below:

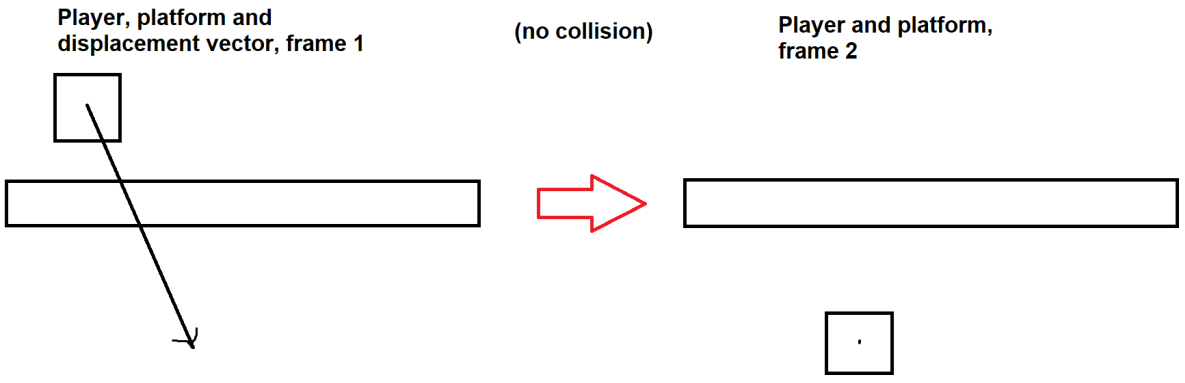
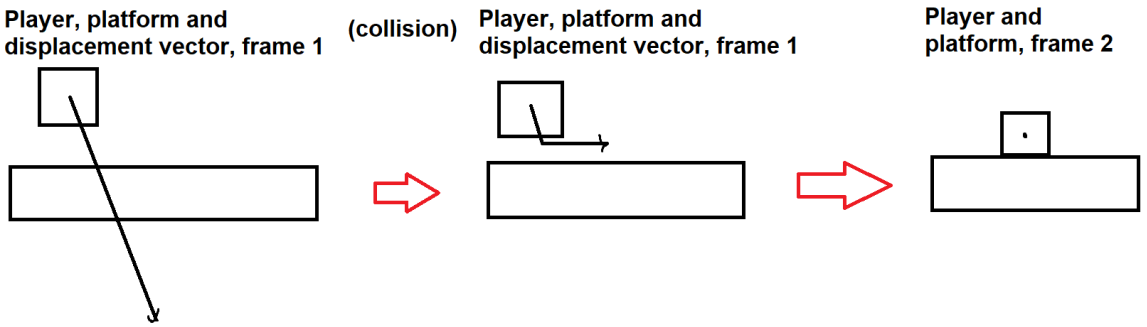


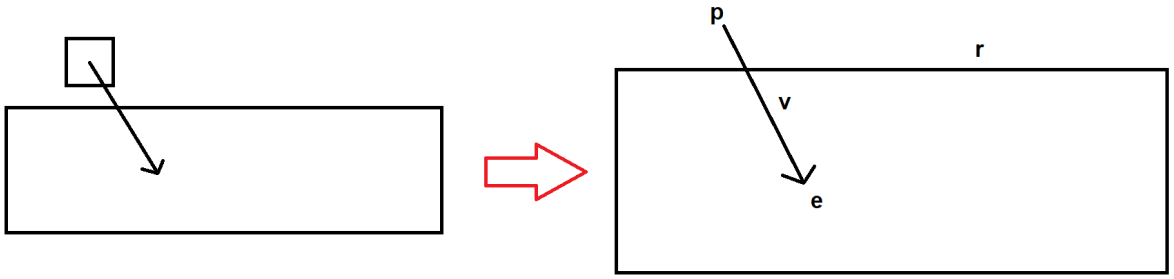
Fig 8.1

But, the necessary information to detect the collision is available, using the displacement vector (Player.dis) as shown below



To achieve this, the Player.collide_with_rectangle method must be changed.
The technique of simplifying the problem to a collision between a rectangle and point is still used.

```
"""
two rectangles colliding can be simplified to a rectangle and point
colliding
"""
```



I use these letters to make the code cleaner and because these are the letters I used when working this out on paper.

```
# combined rectangle
r = rect.combine_rectangle(self.rect)
# point
p = self.rect.pos
# vector
v = self.dis
```

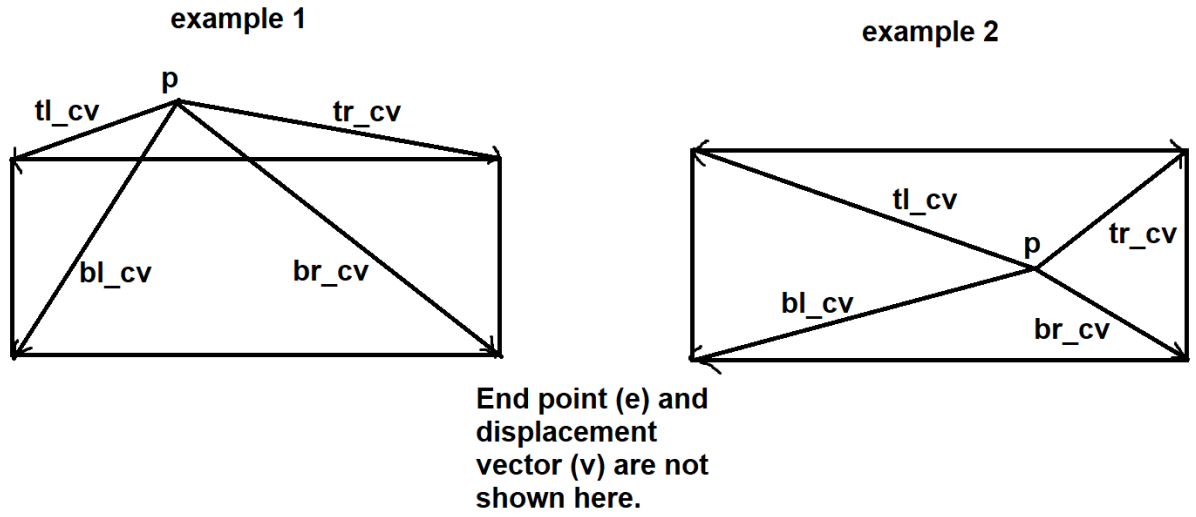
Where the player will end up if there is no collision

```
# end point
e = p + v

# positions of the sides of the rectangle
right = r.right_pos()
bottom = r.bottom_pos()
left = r.left_pos()
top = r.top_pos()
```

If both the end point and the current position are on the same side of the rectangle, there can't be a collision, so the method can return early.

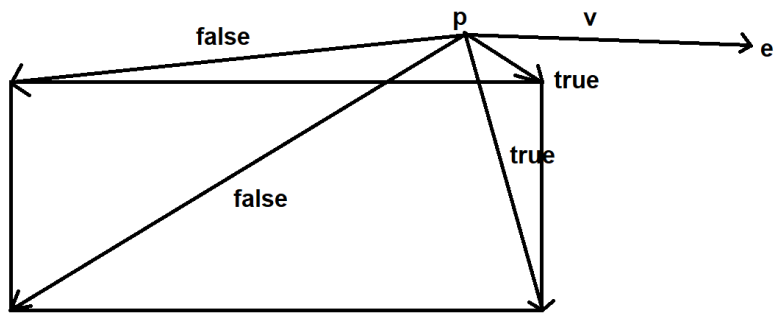
```
if (
    (p.x > right and e.x > right)
    or (p.y > bottom and e.y > bottom)
    or (p.x < left and e.x < left)
    or (p.y < top and e.y < top)
):
    return
```

Bottom right corner vector, top left corner vector, etc.




```
# vectors from p to each corner of r
br_cv = r.bottom_right_pos() - p
bl_cv = r.bottom_left_pos() - p
tr_cv = r.top_right_pos() - p
tl_cv = r.top_left_pos() - p
```

The dot product of two vectors is positive if the angle between then is less than 90 degrees, it is also a lot neater and computationally faster than working out the angle between the vectors directly




```
# sign (+/-) of dot product of corner vector with v
dp_br_sign = Vector.dot(v, br_cv) > 0.0
dp_bl_sign = Vector.dot(v, bl_cv) > 0.0
```

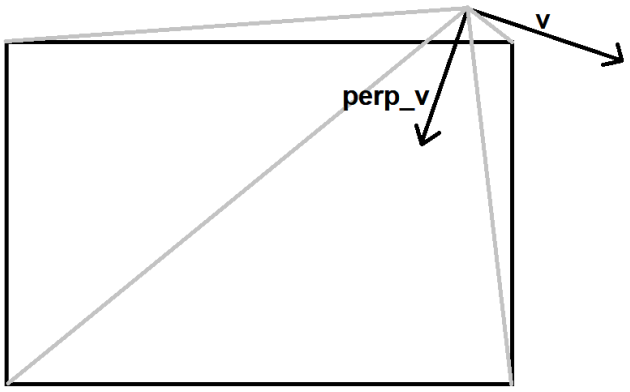
```
dp_tl_sign = Vector.dot(v, tl_cv) > 0.0
dp_tr_sign = Vector.dot(v, tr_cv) > 0.0
```

This return case may be covered by the first return case.

```
if (
    (not dp_br_sign)
    and (not dp_bl_sign)
    and (not dp_tl_sign)
    and (not dp_tr_sign)
):
    return
```

This stands for perpendicular.

```
# v rotated 90 degrees
perp_v = v.rotate90()

# sign (+/-) of dot product of corner vector with v rotated 90 degrees
perp_dp_br_sign = Vector.dot(perp_v, br_cv) > 0.0
perp_dp_bl_sign = Vector.dot(perp_v, bl_cv) > 0.0
perp_dp_tl_sign = Vector.dot(perp_v, tl_cv) > 0.0
perp_dp_tr_sign = Vector.dot(perp_v, tr_cv) > 0.0
```



This early return case determines whether or not all the corner vectors are all on the same side of the displacement vector, if they are, there can't be a collision.

```
if (
    perp_dp_br_sign and perp_dp_bl_sign and perp_dp_tl_sign and
perp_dp_tr_sign
) or (
    (not perp_dp_br_sign)
    and (not perp_dp_bl_sign)
    and (not perp_dp_tl_sign)
    and (not perp_dp_tr_sign)
):
    return
```

This is the collision response.

no_bounce_collision_x and y are methods which respond to the collision, they do not detect a collision, so if used when there has not been a collision, some weird behaviour can occur. It is called no_bounce_collision, because at some point I may want the player to bounce off a surface.
All the if statements determine which side of the rectangle the player has collided with.

```
if v.x > 0.0:
    if v.y > 0.0:
        if perp_dp_tl_sign:
            # top
            self.no_bounce_collision_y(top)
        else:
            # left
            self.no_bounce_collision_x(left)
    else:
        if perp_dp_bl_sign:
            # left
            self.no_bounce_collision_x(left)
        else:
            # bottom
            self.no_bounce_collision_y(bottom)
else:
    if v.y > 0.0:
        if perp_dp_tr_sign:
            # right
            self.no_bounce_collision_x(right)
        else:
            # top
            self.no_bounce_collision_y(top)
    else:
        if perp_dp_br_sign:
            # bottom
            self.no_bounce_collision_y(bottom)
        else:
            # right
            self.no_bounce_collision_x(right)
```

These are the collision response methods in the Player class.
The centre of the player collides with an infinitely long horizontal or vertical line. The components of the player's velocity that caused the collision is set to 0, its position is set be where the line is and the displacement is set to the movement the player should make in the next frame.

```
# for collisions with the centre of the rectangle
def no_bounce_collision_x(self, x):
    self.vel.x = 0
    self.dis = Vector(
```

```
            (x - self.rect.pos.x) * (self.dis.y / self.dis.x),
            self.dis.y,
        )

    # for collisions with the centre of the rectangle
    def no_bounce_collision_y(self, y):
        self.vel.y = 0
        self.dis = Vector(
            (y - self.rect.pos.y) * (self.dis.x / self.dis.y),
            # self.dis.x,
            y - self.rect.pos.y,
        )
```

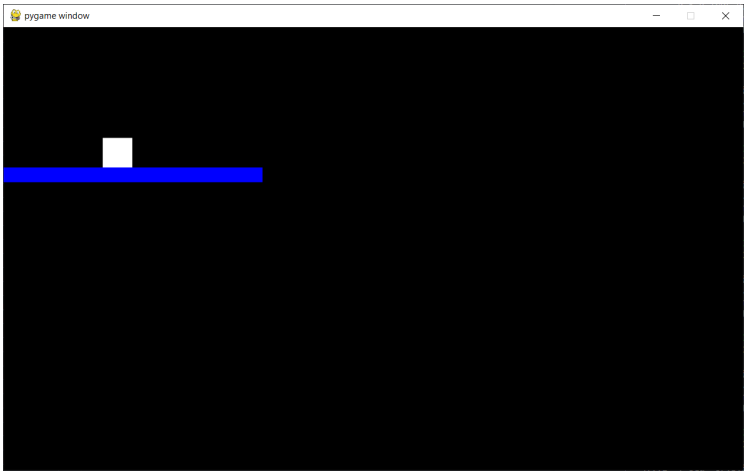
The other collision method is kept and renamed to collide_with_rectangle_discrete, because it uses discrete collision, and the new method uses continuous collision.

The game loop must also be changed to this because the position and displacement before updating the position is needed for the collision method.

```
player.reset_acc()

# acceleration
player.gravity(100)

# update displacement and velocity
player.update_displacement_and_velocity(dt)

# collisions
player.collide_with_rectangle(platform)

# update position
player.update_position()
```

Output:

The player does not slide along the ground. (Logic Error)



To fix this, I log the values from the collision response code.

```
# for collisions with the centre of the rectangle
def no_bounce_collision_y(self, y):
    self.vel.y = 0
    print(y - self.rect.pos.y)
    print(self.dis.x / self.dis.y)
    print()
    self.dis = Vector(
        (y - self.rect.pos.y) * (self.dis.x / self.dis.y),
        y - self.rect.pos.y,
    )
```

Output:
1.1118000000000166
0.6451612903225814

0.0
125.00000000000001

0.0
117.64705882352942

0.0
117.64705882352942

…

This means the displacement is always 0 because

```
y - self.rect.pos.y
```

is 0, so

```
(y - self.rect.pos.y) * (self.dis.x / self.dis.y)
```

is 0.

Fig 8.2

This is changed to

```
# for collisions with the centre of the rectangle
def no_bounce_collision_x(self, x):
    self.vel.x = 0
    self.dis = Vector(
        x - self.rect.pos.x,
        self.dis.y,
    )

# for collisions with the centre of the rectangle
def no_bounce_collision_y(self, y):
```

```
    self.vel.y = 0
    self.dis = Vector(
        self.dis.x,
        y - self.rect.pos.y,
    )
```

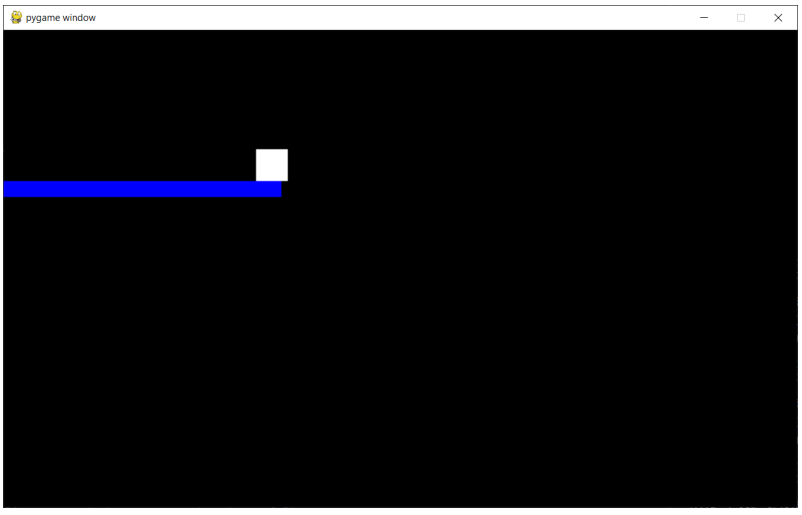
One of the exit clauses is also changed because with this code,

```
if (
    (p.x > right and e.x > right)
    or (p.y > bottom and e.y > bottom)
    or (p.x < left and e.x < left)
    or (p.y < top and e.y < top)
):
    return
```

this return clause will be passed even if there was a collision last frame because the next frame's position will be on the border of the rectangle, which is not covered by this clause. So it is changed to this

```
if (
    (p.x >= right and e.x >= right)
    or (p.y >= bottom and e.y >= bottom)
    or (p.x <= left and e.x <= left)
    or (p.y <= top and e.y <= top)
):
    return
```

Output:



Test:

To test the collision some more, I will add another platform to act as a wall.
(I also changed the position of the other platform to have more space.)
I change the platform variable to a list of platforms

```
platforms = [
    Rectangle(150, 500, 1000, 10),
    Rectangle(400, 500, 10, 500),
]
```

And I change the game loop to loop over all the platforms in that list with this. I will use iteration here because I need to run the same code for each of all the platforms in the level.

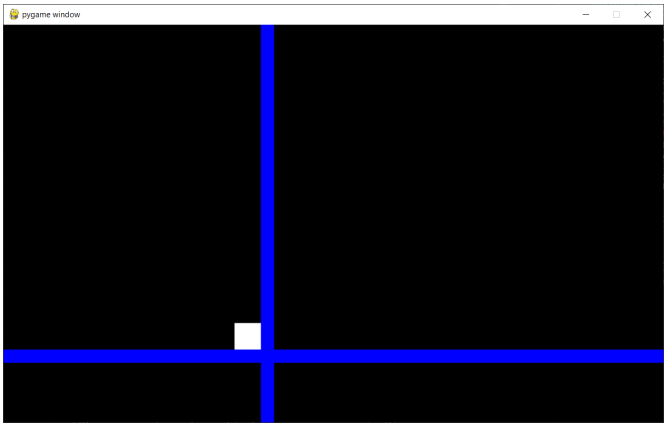
```
for platform in platforms:
    player.collide_with_rectangle(platform)
```

and

```
for platform in platforms:
    pygame.draw.rect(screen, BLUE, platform.corner_rect_tuple())
```

Output:

The player collides with the wall successfully.



Test number	Test Description	Test data (if applicable)	Justification	Outcome
8.1	Does the discrete collision process still work when used in conjunction with a low framerate and thin platform?	The low framerate e.g. 10 (boundary)	If the collision doesn't work, the game will run differently on low-end machines that can't run the game at a high frame rate. This is not ideal.	Failed and fixed by changing to a continuous collision detection and response method (**Fig 8.1**)
8.2	Does the player slide along the ground like it is supposed to when using the continuous collision method?	n/a	If the player does not slide, acceleration will not be able to build up when the player moves intentionally.	Failed and fixed by changing the collision response methods. (**Fig 8.2**)
8.3	Does the collision work horizontally and when in a corner (moving into a wall)?	The new platform at an arbitrary position (valid).	The platforms must act as barriers to get over as well as traditional platforms to provide an obstacle for the player.	As expected

Basic player movement around a level - Prototype 5, Adding Horizontal Movement

To add some controls to the player, I can add a method to the player that applies acceleration to the player when some keys are pressed.
A method on the player class use to move horizontally

```
def move_horizontal(self, left_input, right_input, speed):
    self.apply_acc(Vector((right_input - left_input) * speed, 0))
```

And this is added to the game loop after applying gravity. Pressing a moves the player left, and pressing d moves the player right.

```
player.move_horizontal(left_input=keys[pygame.K_a],
right_input=keys[pygame.K_d], speed=300)
```

Output:
The player slides along the platform when the keys are pressed, there is no friction between the player and the platform.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
9.1	Do the player's horizontal v bn movement inputs make the player move correctly?	The keys the player pressed (A and D) (valid).	The player must be able to move horizontally to move around and complete the level.	As expected
9.2	Do the player's horizontal movement inputs make the player move correctly?	The wrong keys e.g. J and L (erroneous)	The player must not move if the wrong keys are pressed.	As expected

Basic player movement around a level - Prototype 6, Jumping

To implement a jumping mechanism, the player class needs to "know" whether or not the player is grounded. The player is grounded if it has collided with the top of a platform in that frame. In all other cases it is not grounded, so I add this to the player initialisation method

```
self.grounded = False
```

And I add this method

```
def reset_grounded(self):
    self.grounded = False
```

and this method

```
def set_grounded(self):
    self.grounded = True
```

to the player class.

After all the player movement, but before the player collisions, I will call the reset_grounded method on the player.

```
...

# update displacement and velocity
player.update_displacement_and_velocity(dt)

player.reset_grounded()

# collisions
for p in platforms:
    player.collide_with_rectangle(p)

# update position
player.update_position()

...
```

And in the collision method, I will set Player.grounded to true when the player collides with the top of a platform.

```
if v.x > 0.0:
    if v.y > 0.0:
        if perp_dp_tl_sign:
            # top
            self.no_bounce_collision_y(top)
            self.set_grounded()
        else:
```

```
            # left
            self.no_bounce_collision_x(left)
    else:
        if perp_dp_bl_sign:
            # left
            self.no_bounce_collision_x(left)
        else:
            # bottom
            self.no_bounce_collision_y(bottom)
else:
    if v.y > 0.0:
        if perp_dp_tr_sign:
            # right
            self.no_bounce_collision_x(right)
        else:
            # top
            self.no_bounce_collision_y(top)
            self.set_grounded()
    else:
        if perp_dp_br_sign:
            # bottom
            self.no_bounce_collision_y(bottom)
        else:
            # right
            self.no_bounce_collision_x(right)
```

To test this grounded attribute, I will log the player.grounded variable each frame (in the game loop) after the collision method calls.

```
...

player.reset_grounded()

# collisions
for platform in platforms:
    player.collide_with_rectangle(platform)

print(player.grounded)

...
```

Output:
The stream of logging changes from true to false when the player hits the ground.
…
False

False
False
False
False
False
False
False
True
True
True
True
True
True
True
…

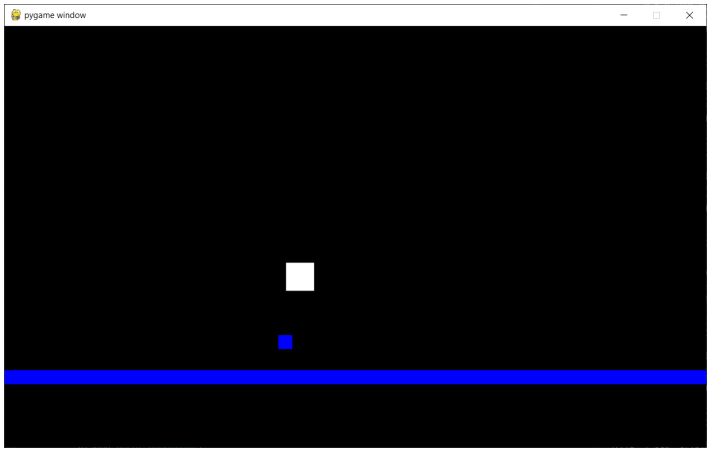
To add a jump method, an acceleration will be applied to the player when a key is pressed and the player is grounded.
This is the method in the player class

```
def jump(self, jump_input, power):
    # if the user is pressing jump and the player can jump
    if jump_input and self.grounded:
        self.apply_acc(Vector(0, -power))
```

And this is added to the game loop

```
player.jump(jump_input=keys[pygame.K_SPACE], power=4000)
```

Output:
The player can jump over the platform.



Test number	Test Description	Test data (if applicable)	Justification	Outcome
10.1	Can the player jump to clear an obstacle?	The user pressing space. (valid)	The player must be able to jump to overcome obstacles.	As expected

Basic player movement around a level - Prototype 7, Adding friction and air resistance

The player does not slow down when not pressing the movement keys, so air resistance and friction from the ground must be added.
Methods in the player class

```
def air_resistance(self, power):
    self.apply_acc(self.vel.flip() * power)

def ground_friction(self, power):
    if self.grounded:
        self.apply_acc(Vector(-self.vel.x * power, 0))
```

Method calls in the game loop

```
player.air_resistance(power=2)
player.ground_friction(power=5)
```

Output:
The player now slows to a stop after releasing the movement keys.

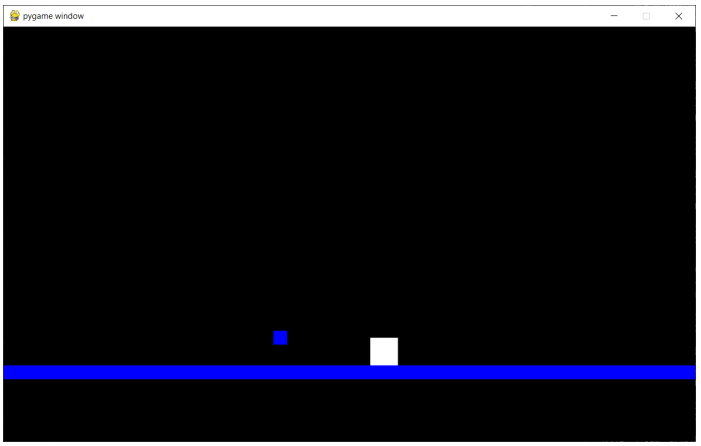


Fig 11.1
But now, the player moves faster in the air than the ground because air resistance is less than ground friction, so less acceleration should be applied when moving in the air.
This is the new horizontal movement in the player class

```
def move_horizontal(
    self, left_input, right_input, grounded_speed, ungrounded_speed
):
    # use the movement inputs to calculator an acceleration to be
applied
    self.apply_acc(
        Vector(
            (right_input - left_input)
            * (grounded_speed if self.grounded else ungrounded_speed),
            0,
```

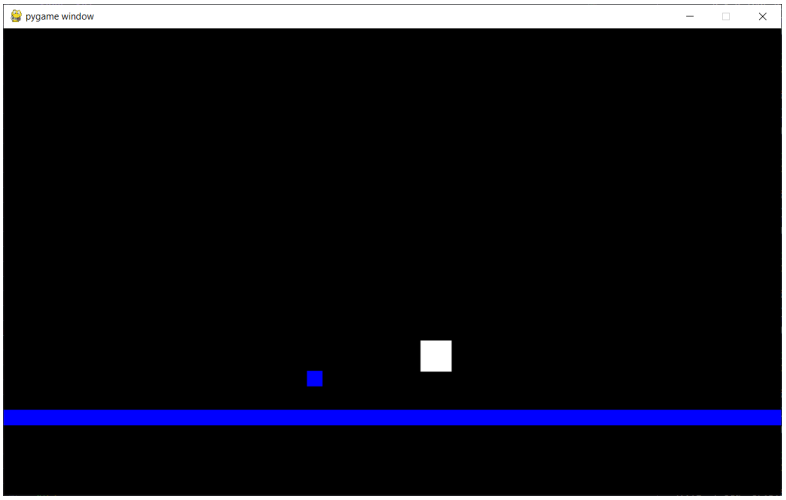
```
            )
        )
```

And the method call in the game loop is changed to

```
player.move_horizontal(left_input=keys[pygame.K_a],
right_input=keys[pygame.K_d], grounded_speed=2000,
ungrounded_speed=1000)
```

Output:

The player moves more normally now



Test number	Test Description	Test data (if applicable)	Justification	Outcome
11.1	Does the player gradually slow down while moving due to the air resistance and fraction added?	The player's movement (applied by the user). (valid)	The player must not slide forever after picking up speed. This will make the game easier for beginners and more intuitive.	As expected
11.2	Does the player move correctly while in the air?	The player's movement (applied by the user). (valid)	The player must have more control on the ground than in the air. This is achieved by a slower movement acceleration but less resistance to motion in the air. This will make the game's movement more intuitive.	Failed and fixed by changing the player's speed in the air. (**Fig 11.1**)

.

Camera

When the size of the screen changes, the size of the player and platforms in pixels stays the same which looks bad.

Test:

```
WIDTH = 300
HEIGHT = 200
```

I also changed the location of the platforms to fit on the screen.

Output:

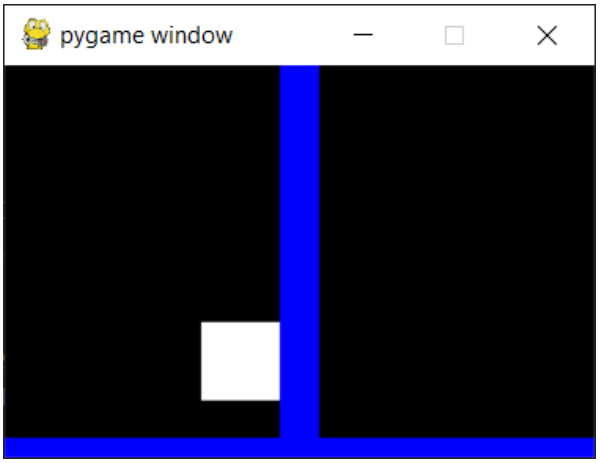


Fig 12.1

To fix this, the widths and heights and positions of all the rectangles could be scaled by the width and the height of the screen in pixels, but a better solution is to use a Camera class which has methods to transform the coordinates of the game objects in game space to positions in pixel space to display on the screen.

The camera class should have the width and height of the screen in pixel, the position of the camera in the game and the zoom/field of view/how much of the game environment the camera can see. It should also be able to move to follow the player.

```
class Camera:
    def __init__(self, x, y, zoom, width, height):
        self.pos = Vector(x, y)
        self.zoom = zoom
        self.width = width
        self.height = height
```

These are the methods for transforming x and y coordinates

```
    # convert a position in the x axis
    def game_space_to_screen_space_x_pos(self, x):
        return (x - self.pos.x) * self.zoom + self.screen_width_pix / 2
    # convert a position in the y axis
    def game_space_to_screen_space_y_pos(self, y):
```

```
        return (y - self.pos.y) * self.zoom + self.screen_height_pix /
2
```

These are the methods for transforming x and y dimensions

```
    # convert a dimension in the x axis
    def game_space_to_screen_space_x_dim(self, x):
        return x * self.zoom
    # convert a dimension in the y axis
    def game_space_to_screen_space_y_dim(self, y):
        return y * self.zoom
```

These methods combine the other methods to transform position and dimension vectors

```
    # convert a position
    def game_space_to_screen_space_pos(self, pos):
        return Vector(
            self.game_space_to_screen_space_x_pos(pos.x),
            self.game_space_to_screen_space_y_pos(pos.y),
        )
    # convert a dimension
    def game_space_to_screen_space_dim(self, dim):
        return Vector(
            self.game_space_to_screen_space_x_dim(dim.x),
            self.game_space_to_screen_space_y_dim(dim.y),
        )
```

And this method transforms a rectangle tuple from the Rectangle.corner_rect_tuple method

```
    def game_space_to_screen_space_corner_rect_tuple(self, rect_tuple):
        return (
            self.game_space_to_screen_space_x_pos(rect_tuple[0]),
            self.game_space_to_screen_space_y_pos(rect_tuple[1]),
            self.game_space_to_screen_space_x_dim(rect_tuple[2]),
            self.game_space_to_screen_space_y_dim(rect_tuple[3]),
        )
```

Now, in the game loop, when displaying a rectangle, the program first converts it using the camera created here

```
camera = Camera(0, -10, 40, WIDTH, HEIGHT)
```

To move the camera to the player's position, so the player is always in the centre of the screen

```
camera.pos = player.rect.pos
```

And to display the payer (for example)

```
pygame.draw.rect(
        screen,
        WHITE,
        camera.game_space_to_screen_space_corner_rect_tuple(
            player.rect.corner_rect_tuple()
        ),
```

```
    )
```

Output:
The zoom works, and the camera follows the player, but it does not have the ability to scale the output to the size of the screen. (Logic Error)

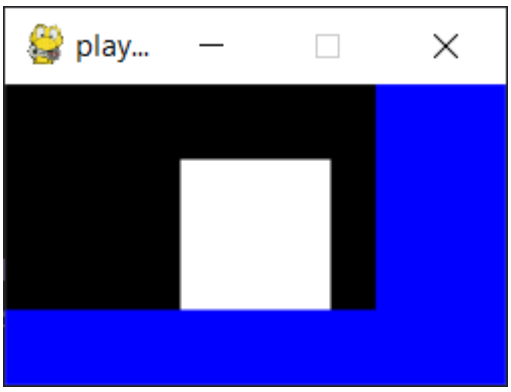


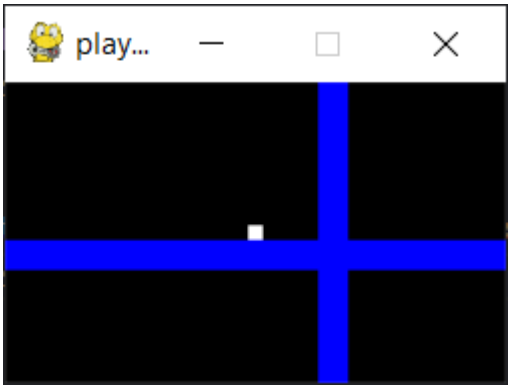
Fig 12.2
To fix this, the zoom of the camera must be calculated using the size of the screen in the camera's initialisation method.

```
ef __init__(self, x, y, screen_width_pix, screen_height_pix,
game_space_visible_width=0, game_space_visible_height=0):
        self.screen_width_pix = screen_width_pix
        self.screen_height_pix = screen_height_pix
        self.pos = Vector(x, y)
        # If the visible height of the screen is given, use that to
calculate the zoom
        if game_space_visible_height == 0:
            self.zoom = self.screen_width_pix /
game_space_visible_width
        # same for the width
        elif game_space_visible_width == 0:
            self.zoom = self.screen_height_pix /
game_space_visible_height
```

Now the camera is created like this

```
camera = Camera(0, 5, WIDTH, HEIGHT, game_space_visible_height=20)
```

Output:



However, if neither screen dimension argument is given

```
camera = Camera(0, 5, WIDTH, HEIGHT)
```

Output:
The program crashes (Division by 0 zero)

```
  File "D:\usb for win down backup (goat)\GOATwvenv\game\game.py", line 28, in main
    camera = Camera(0, 5, WIDTH, HEIGHT)
  File "D:\usb for win down backup (goat)\GOATwvenv\game\camera.py", line 11, in __init__
    self.zoom = self.screen_width_pix / game_space_visible_width
ZeroDivisionError: division by zero
```

Fig 12.3
So the initialisation method is changed to

```
def __init__(self, x, y, screen_width_pix, screen_height_pix,
game_space_visible_width=0, game_space_visible_height=0):
        self.screen_width_pix = screen_width_pix
        self.screen_height_pix = screen_height_pix
        self.pos = Vector(x, y)
        # If the visible height of the screen is given, use that to
calculate the zoom
        if game_space_visible_width != 0:
            self.zoom = self.screen_width_pix /
game_space_visible_width
        # same for the width
        elif game_space_visible_height != 0:
            self.zoom = self.screen_height_pix /
game_space_visible_height
```

So the program will only divide if the divisor is not zero.

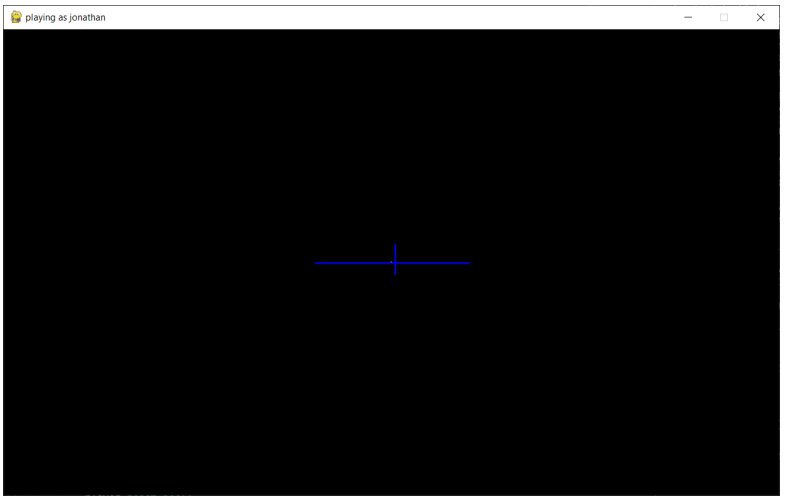
Output:
If no dimension is given, Camera.zoom is not created, so the program crashes when it is used. (Logic Error)

```
  File "D:\usb for win down backup (goat)\GOATwvenv\game\camera.py", line 18, in
game_space_to_screen_space_x_pos
    return (x - self.pos.x) * self.zoom + self.screen_width_pix / 2
AttributeError: 'Camera' object has no attribute 'zoom'
```

So a default value for zoom must be added along with a message that zoom cannot be calculated.

```
else:
    print("Camera zoom cannot be calculated.")
    self.zoom = 1
```

Output:

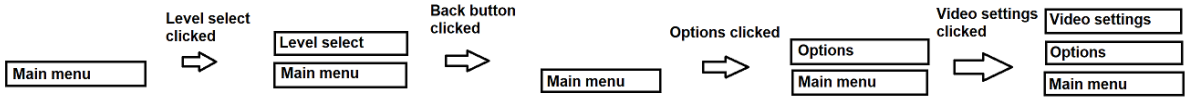


Camera zoom cannot be calculated.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
12.1	Can the screen zoom in and out?	The amount of zoom applied e.g. 40 (valid)	The screen must be able to zoom in and out to accommodate for different resolutions.	Failed and fixed by adding camera zoom logic (**Fig 12.1**)
12.2	Does the zoom take into account the screen resolution?	The screen resolution e.g. 300x200 (valid)	The user must be able to see the same amount of the level on all different resolutions.	Failed and fixed by changing the way the zoom is calculated. (**Fig 12.2**)
12.3	Does the camera always calculate zoom correctly?	The requested visible width and height (0x0) (boundary).	The program must not crash depending on the inputs to the visual parameters.	Failed and fixed by changing the selection statements for the calculation of the zoom to take into account the edge case. (**Fig 12.3**)

Menus - Protype 1, Navigating between two screens

The game will have a main menu and a level select screen.
For this, I will use a stack-like (FILO) data structure with the main menu at the bottom, when the user selects the options screen or level select screen from that, it will be pushed to the stack. Then when the user selects the video setting, that screen will be pushed to the stack and so on. When the user clicks back, the current screen will be popped from the stack and the screen will return to the lower one on the stack.



To create a simple prototype, the user will start on a screen that displays nothing, then then they press A or B and that brings them to a screen with that letter, if on screen A, the user can click 1, 2, or 3 and that will bring up a screen with that number on it. The escape key will be used as the back button, and I will log the menu stack after the screen has changed.

To start, I create the beginning/base/root screen which will be the screen that displays nothing.

```
import pygame



def root():

    WIDTH = 1000
    HEIGHT = 600

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    clock = pygame.time.Clock()

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001
```

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_a:
                    # go to screen A
                    pass
                elif event.key == pygame.K_b:
                    # go to screen b
                    pass

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        pygame.display.flip()

    pygame.quit()


if __name__ == "__main__":
    root()
```

Output:

Then I create a screen for screen A. I use code from this web page to display text
https://stackoverflow.com/questions/20842801/how-to-display-text-in-pygame

```
def a():
    WIDTH = 1000
    HEIGHT = 600

    FPS = 60
```

```
    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    pygame.init()

    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    clock = pygame.time.Clock()

    pygame.font.init()
    # create a font
    my_font = pygame.font.SysFont('Monospace', 30)
    # render some text
    text_surface = my_font.render('A', False, (0, 0, 0))

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN:
                pass

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        # draw the text to the screen
        screen.blit(text_surface, (0, 0))

        pygame.display.flip()

    pygame.quit()
```

Output:
No text is displayed. (Logic Error)

Fig 13.1

This happens because the text is black. So, I will the text creation code to this.

```
# create a font
my_font = pygame.font.SysFont('Comic Sans MS', 200)
# render some text
text_surface = my_font.render('A', False, WHITE)
```

Output:

Now I change the root screen to run the A screen when A is pressed.

```
if event.key == pygame.K_a:
    # go to screen A
    a()
```

Output:

The screen changes to A when A is pressed.

After A is pressed:

To go back, I add this to the screen A event loop

```
if event.type == pygame.KEYDOWN:
    # go back if Esc clicked
    if event.key == pygame.K_ESCAPE:
        running = False
```

Output:

File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\menuprototype.py", line 83, in root
keys = pygame.key.get_pressed()
pygame.error: video system not initialized
(Logic Error)

Fig 13.2

This happens because when the escape key is pressed, running is set to false, and pygame quits, so there is no screen for root to work on.
To fix this, the screen will be created separately in a main function and will be passed into the root and a functions.

```
import pygame


def a(screen):

    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
```

```
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    # create a font
    my_font = pygame.font.SysFont("Comic Sans MS", 200)
    # render some text
    text_surface = my_font.render("A", False, WHITE)

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN:
                # go back if Esc clicked
                if event.key == pygame.K_ESCAPE:
                    running = False

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        # draw the text to the screen
        screen.blit(text_surface, (0, 0))

        pygame.display.flip()


def root(screen):

    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
```

```
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_a:
                    # go to A, passing along the screen
                    a(screen)
                elif event.key == pygame.K_b:
                    # go to screen b
                    pass

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        pygame.display.flip()


def main():

    WIDTH = 1000
    HEIGHT = 600

    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))

    pygame.font.init()

    # go to root, allowing it to use the screen
    root(screen)

    # this will only run when all the loop in the root screen has
finished
```

```
    pygame.quit()


if __name__ == "__main__":
    main()
```

Output:

However, if the cross is clicked, the screen only goes back, it doesn't quit pygame. (Logic Error)

Fig 13.3

So, if the cross is clicked, the function should return true and pass this back, quitting out of each screen until pygame is quit.

To do this, the pygame.QUIT event is changed to this

```
if event.type == pygame.QUIT:
    return True
```

And the screen function calls are changed to this

```
if a(screen):
    return True
```

Output:

Pygame quits when the cross is clicked.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
13.1	Does the empty pygame loop for the menu create a blank screen that quits when the cross is clicked?	n/a	The screen I need a basis for the menu system to build upon.	As expected
13.2	Does the sample text show up on the screen?	The sample text, colour and font ("A", BLACK, Comic Sans MS) (Valid)	The menu screens must display text to show the user what screen they are on and how to interact with the manu to achieve their desired outcome.	Failed and fixed by changing the colour of the text. (**Fig 13.1**)

13.3	Is the prototype menu screen shown when a key is pressed?	The key pressed, “A”. (valid)	The program must be able to switch screens by calling the function corresponding to a menu screen within the parent screen’s function.	As expected
13.4	Can I exit the current screen and return to the parent screen by pressed escape	The key pressed (Esc), (valid)	The user must be able to return to the previous screen for fully functional navigation.	Failed and fixed by creating the pygame screen instance outside the menu functions and passing a reference to it between the functions. (**Fig 13.2**)
13.5	Does the cross exit out of all of the screen all the way to the root screen and then quit out of pygame?	n/a	The user should be able to completely exit the game without manually clicking through all the previous screens.	Failed and fixed by adding a boolean return value to the screen functions. To tell the parent screen whether to continue exiting the screens or stay on the parent screen after exiting the child screen. (**Fig 13.3**)

Menus - Protype 2, A button in pygame

To make a menu screen, I need to make a buttons class to store data about the button and define its behaviour. The OO aspect of this design helps because it can encapsulate the button's data, e.g. the string of button text and the methods, like what happens when the button is pressed.

```
from geometry.rectangle import Rectangle

import pygame

class Button:
    def __init__(self, text, x, y, hw, hh, on_click, font, font_col):
        self.text = text
        # the shape the button will be
        self.rect = Rectangle(x, y, hw, hh)
        # a function that is called when the button is clicked
        self.on_click = on_click
        # the text that will be displayed
        self.text_render = font.render(self.text, False, font_col)
        # whether the button is currently being pressed or not
        self.pressed = False

    def draw(self, screen):
        # draw the button's rectangle
        pygame.draw.rect(screen, (126, 126, 126),
self.rect.corner_rect_tuple())
        # draw the button's text
        screen.blit(self.text)

    def check_mouse_down(self, mouse_pos):
        # if the mouse is pressed, and it is in the button's rectangle,
update the pressed variable
        if self.rect.contains_point(mouse_pos):
            self.pressed = True

    def check_mouse_up(self, mouse_pos):
        # if the mouse is released and the button was pressed before
and the mouse is in the button's rectangle, call the given on_click
function
        if self.pressed and self.rect.contains_point(mouse_pos):
            self.on_click()
        # the button should never be pressed right after the mouse is
released
```

```
        self.pressed = False
```

I will rename the root function to main_menu, change the trigger for the A screen from a key press to the function run when the button is clicked and draw the button each frame.

```
def main_menu(screen):

    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    a_button = Button(
        "A",
        WIDTH / 2,
        HEIGHT / 2,
        100,
        100,
        a,
        pygame.font.SysFont("Comic Sans MS", 50),
        WHITE,
    )

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        mouse_pos_tuple = pygame.mouse.get_pos()
        mouse_pos = Vector(mouse_pos_tuple[0], mouse_pos_tuple[1])

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return True
            if event.type == pygame.MOUSEBUTTONDOWN:
                a_button.check_mouse_down(mouse_pos)
```

```
            if event.type == pygame.MOUSEBUTTONUP:
                a_button.check_mouse_up(mouse_pos)

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        a_button.draw(screen)

        pygame.display.flip()
```

Output:

File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\button.py", line 21, in draw
screen.blit(self.text)
TypeError: argument 1 must be pygame.Surface, not str
(Logic Error)

Fig 14.1

I change the draw method to this to draw the text render instead of the text

```
def draw(self, screen):
    # draw the button's rectangle
    pygame.draw.rect(screen, (126, 126, 126),
self.rect.corner_rect_tuple())
    # draw the button's text
    screen.blit(self.text_render)
```

Output:

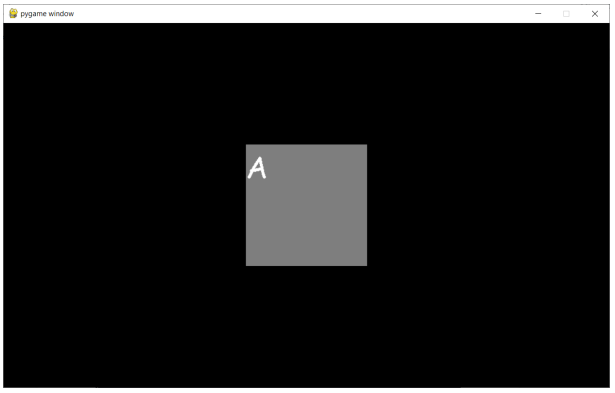
File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\button.py", line 21, in draw
screen.blit(self.text_render)
TypeError: function missing required argument 'dest' (pos 2)
(Logic Error)

Fig 14.2

The screen.blit call needs a position to draw the text to, like this

```
screen.blit(self.text_render, (self.rect.left_pos(),
self.rect.top_pos()))
```

Output:



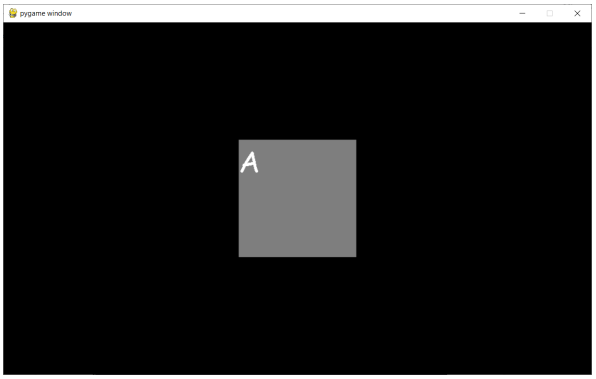
Test:
Clicking the button

Output:
Then, when clicking the button
  File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\button.py", line 31, in check_mouse_up
    self.on_click()
TypeError: a() missing 1 required positional argument: 'screen'
(Logic Error)

Fig 14.3
This means I need to create a function that takes no arguments and calls a(screen) and use that function for the button's on_click function instead.

```
def a_button_on_click():
    a(screen)

a_button = Button(
    "A",
    WIDTH / 2,
    HEIGHT / 2,
    100,
    100,
    a_button_on_click,
    pygame.font.SysFont("Comic Sans MS", 50),
    WHITE,
)
```

Output:
The button works as expected.



Test:
Clicking the cross (quit) while on the "A" screen (child screen).

Output:

When quitting on the A screen, it only goes back to the main menu screen. (Logic Error)

Fig 14.4

This is because I do not return the boolean value (whether the user pressed back (esc) or quit (cross)) from when the button calls the A screen function.
To fix this, I return the value back down the stack of function calls like this
When creating the button

```
def a_button_on_click():
    return a(screen)

a_button = Button(
    "A",
    WIDTH / 2,
    HEIGHT / 2,
    100,
    100,
    a_button_on_click,
    pygame.font.SysFont("Comic Sans MS", 50),
    WHITE,
)
```

When the button is pressed

```
def check_mouse_up(self, mouse_pos):
    # if the mouse is released and the button was pressed before and
the mouse is in the button's rectangle, call the given on_click
function
    if self.pressed and self.rect.contains_point(mouse_pos):
        return self.on_click()
    # the button should never be pressed right after the mouse is
released
    self.pressed = False
```

And from the main menu screen's event loop when it checks if the button has been pressed

```
if event.type == pygame.MOUSEBUTTONUP:
    if a_button.check_mouse_up(mouse_pos):
        return True
```

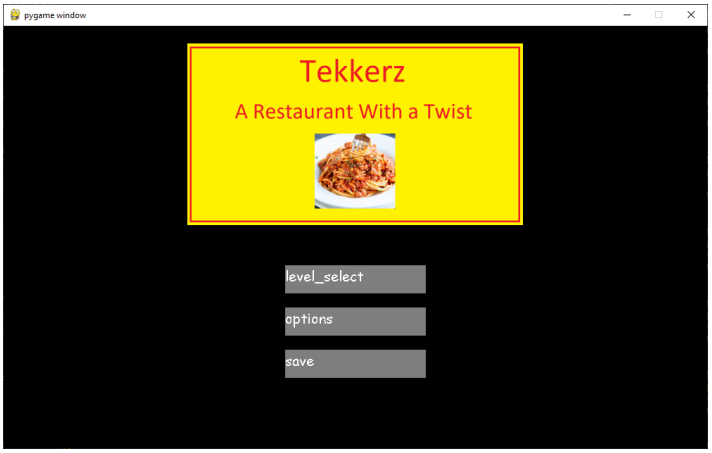
I will also add the Tekkerz logo here by loading the image into man memory like this outside the game loop

```
logo_image = pygame.image.load("assets/logosmall.png")
```

and then I'll add this to draw the image onto the screen

```
screen.blit(logo_image, (WIDTH / 2 - logo_image.get_width() / 2, 25))
```

Output:



Test number	Test Description	Test data (if applicable)	Justification	Outcome
14.1	Does the button display properly?	The rectangle and text of the button “A” (valid)	The buttons must be visible for the user to use it.	Failed and fixed by providing a surface and position to draw the button on (**Fig 14.1**, **Fig 14.2**)
14.2	Does clicking the button take the user to the associated screen?	n/a	The user must be able to click buttons to navigate the menu screens.	Failed and fixed by creating a button on_click function that requires no arguments. The definition of this function provides the arguments to the child screen’s function. (**Fig 14.3**)
14.3	Does clicking cross exit the user from all the screens from a screen accessed by a button?	n/a	The user should be able to exit the game in one go, rather than clicking through all the screens manually.	Failed and fixed by adding and passing return values to the button on click function (**Fig 14.4**)
14.4	Does the main menu load and display the Tekkerz logo	the image file name, “logosmall.png” (valid)	To promote the game Tekkerz, the logo should be shown on the man menu (the user will see this whenever the game loads).	As expected

Menus - Prototype 3, Buttons to access game menus

To evolve this prototype into something more closely resembling the final menus, i will change the main menu screen to this

```
def main_menu(screen):

    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    my_font = pygame.font.SysFont("Comic Sans MS", 50)
    # the function to run when the level select button is clicked
    def level_selct_button_on_click():
        return level_select(screen)
    # creating the level select button
    level_select_button = Button(
        "level_select",
        WIDTH / 2,
        HEIGHT / 2,
        100,
        100,
        level_selct_button_on_click,
        my_font,
        WHITE,
    )
    # the same for the options button
    def options_button_on_click():
        return options(screen)

    options_button = Button(
        "options",
        WIDTH / 2,
        HEIGHT / 2,
        100,
```

```
        100,
        options_button_on_click,
        my_font,
        WHITE,
    )

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        mouse_pos = Vector.from_tuple(pygame.mouse.get_pos())

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return True

            # check all buttons for mouse down
            if event.type == pygame.MOUSEBUTTONDOWN:
                level_select_button.check_mouse_down(mouse_pos)
                options_button.check_mouse_down(mouse_pos)

            # check all buttons for mouse up
            if event.type == pygame.MOUSEBUTTONUP:
                if level_select_button.check_mouse_up(mouse_pos):
                    return True
                if options_button.check_mouse_up(mouse_pos):
                    return True

            if event.type == pygame.KEYDOWN:
                # go back if Esc clicked
                if event.key == pygame.K_ESCAPE:
                    return

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        level_select_button.draw(screen)
        options_button.draw(screen)

        pygame.display.flip()
```

Where level_select is

```
def level_select(screen):
    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    # create some text to show which screen it is
    my_font = pygame.font.SysFont("Comic Sans MS", 50)
    text_render = my_font.render("level_select", False, WHITE)

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        mouse_pos = Vector.from_tuple(pygame.mouse.get_pos())

        for event in pygame.event.get():
            # go all the way back if cross clicked
            if event.type == pygame.QUIT:
                return True
            if event.type == pygame.KEYDOWN:
                # go back if Esc pressed
                if event.key == pygame.K_ESCAPE:
                    return False

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        screen.blit(text_render, (0, 0))

        pygame.display.flip()
```

And options is

```
def options(screen):
```

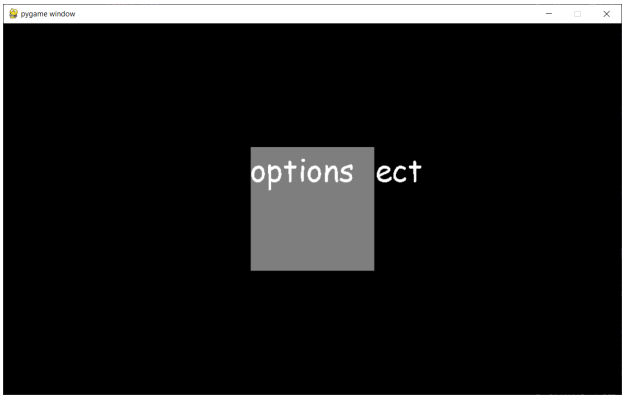
```
    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    # create some text to whow which screen it is
    my_font = pygame.font.SysFont("Comic Sans MS", 50)
    text_render = my_font.render("options", False, WHITE)

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        mouse_pos = Vector.from_tuple(pygame.mouse.get_pos())

        for event in pygame.event.get():
            # go all the way back if cross clicked
            if event.type == pygame.QUIT:
                return True
            if event.type == pygame.KEYDOWN:
                # go back if Esc pressed
                if event.key == pygame.K_ESCAPE:
                    return False

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        screen.blit(text_render, (0, 0))

        pygame.display.flip()
```

Test:

```
WIDTH = 1000
HEIGHT = 600
```

```
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.font.init()

# go to root, allowing it to use the screen
main_menu(screen)

# this will only run when all the loop in the root screen has finished
pygame.quit()
```

Output:



(Logic Error)
The buttons are positioned incorrectly

Fig 15.1
I will change their positions and dimensions with this

```
level_select_button = Button(
    "level_select",
    WIDTH / 2,
    3 * HEIGHT / 10,
    200,
    50,
    level_selct_button_on_click,
    my_font,
    WHITE,
)

def options_button_on_click():
    return options(screen)

options_button = Button(
    "options",
```

```
    WIDTH / 2,
    5 * HEIGHT / 10,
    200,
    50,
    options_button_on_click,
    my_font,
    WHITE,
)
```

Output:

Then when clicking the buttons

The creation of the buttons cam be simplified using inline lambda functions here

```
level_select_button = Button(
    "level_select",
    WIDTH / 2,
    3 * HEIGHT / 10,
    200,
    50,
    lambda : level_select(screen),
    my_font,
    WHITE,
)
```

```
options_button = Button(
    "options",
    WIDTH / 2,
    5 * HEIGHT / 10,
    200,
    50,
    lambda : options(screen),
    my_font,
    WHITE,
)
```

Test number	Test Description	Test data (if applicable)	Justification	Outcome
15.1	Do the buttons display properly and in the right positions?	The positions and dimensions of the buttons along with their text, ("options", WIDTH / 2, 5 * HEIGHT / 10, 200, 50) (valid)	The buttons must be displayed properly for the user to use them easily.	Failed partially and fixed by changing the positions of the buttons. (**Fig 15.1**)

Login - Prototype 1, Storing and Loading some example user data

To login, I need to store some users, each having a username and a password. They will also need to store some information about their progression in the game.
To do this, I create a simple user class

```
class User:
    def __init__(self, username, password):

        self.username = username
        self.password = password
        self.progression = None

    def __str__(self):
        return f'username: {self.username}, password: {self.password},
progression: {self.progression}'
```

Then I create a list of users to store in a file and test the login system on

```
users = [User("jonathan", "asdfgh"), User("Bader", "qwerty")]
```

To save them to a file I use pickle (a serialisation library)

```
from pickle import dump

with open(r"./users.p", "wb") as f:
    dump(users, f)
```

Output (in users.p):

```
��|������]�(�__main__��User���)��}�(�username��jonathan��password��asd
fgh��
progression�Nubh)��}�(h�Bader�h�qwerty�h
Nube.
```

Then to read the file and load the users back into the program, I use this code

```
from pickle import load

with open(r"./users.p", "rb") as f:
    users = load(f)

print(users)
```

Output:
[<__main__.User object at 0x000001CF52A63F40>, <__main__.User object at 0x000001CF52DF0160>]

I want to see that the data in each user object is still the same so I use this

```
print([str(user) for user in users])
```

Output:
['username: jonathan, password: asdfgh, progression: None', 'username: Bader, password: qwerty, progression: None']

To make saving and loading objects to a file simpler, I create these functions

```
import pickle

def load_obj(file_path):
    with open(file_path, r"rb") as f:
        users = pickle.load(f)
    return users

def save_obj(obj, file_path):
    with open(file_path, r"wb") as f:
        pickle.dump(obj, f)
```

So now the code looks like this

```
save_obj(users, r"./users.p")
users = load_obj(r"users.p")
```

Test number	Test Description	Test data (if applicable)	Justification	Outcome
16.1	Creating an example user and serialising it to save in secondary storage. Does an example user object (used to group data)?	The data in the user (username=" Jonathan", password="asdfgh", progression =None) (valid)	The program must correctly save user data so they can quit and login later.	As Expected (in the external file)
16.2	Can the program correctly load user data bak into main memory from secondary storage?	The binary data in the file. (valid)	User data must be loaded correctly so the user can continue where they left off after saving and quitting before.	As expected

Login - Prototype 2, A Login screen that checks the entered username and password

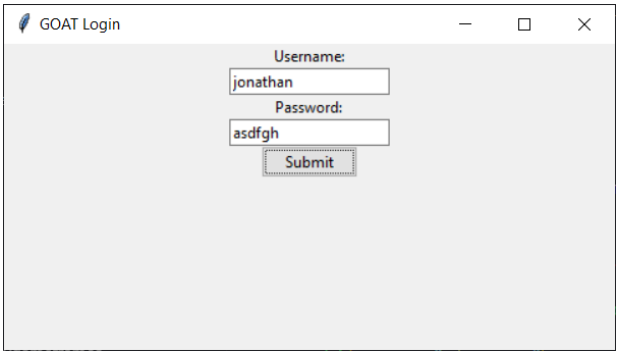
For the user to login, I use tkinter and this boilerplate https://stacktuts.com/tkinter-boiler-plate I will change it to this. I will use iteration a lot in this section because there are many instances where I need to run a test on the details of all the users.

```
import tkinter as tk
from tkinter import ttk

class LoginScreen(tk.Tk):
    def __init__(self, users, on_login):
        super().__init__()

        self.users = users
        self.on_login = on_login

        self.geometry("480x240")
        self.title("GOAT Login")

        ttk.Label(self, text="Username:").pack()
        self.username_entry = ttk.Entry(self)
        self.username_entry.pack()

        ttk.Label(self, text="Password:").pack()
        self.password_entry = ttk.Entry(self)
        self.password_entry.pack()

        ttk.Button(self, text="Submit", command=self.on_submit).pack()

    def on_submit(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        for user in self.users:
            if user.username == username and user.password == password:
                self.on_login(user)
```

And in the main code

```
def on_login(user):
    print("login successful")
    print("user is " + str(user))

LoginScreen(users, on_login).mainloop()
```

Test:



Output:

login successful
user is username: jonathan, password: asdfgh, progression: None

Now, I want the entries to reset after the user clicks submit.

```
def on_submit(self):
    username = self.username_entry.get()
    password = self.password_entry.get()
    # go through all the user and check if any of their details match
with the login details
    for user in self.users:
        if user.username == username and user.password == password:
            self.on_login(user)
    self.username_entry.set("")
    self.username_entry.set("")
```

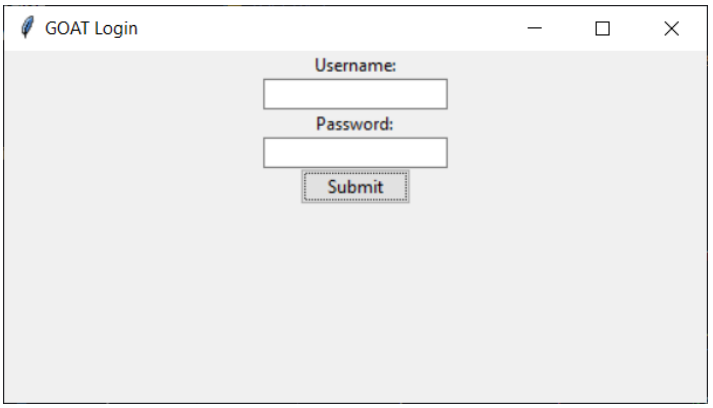
Output:
File "D:\usb for win down backup (goat)\GOATwvenv\tkuiscreens\loginscreen.py", line 32, in on_submit
self.username_entry.set("")
AttributeError: 'Entry' object has no attribute 'set'
(Logic Error)

Fig 17.1
There is no way to set the text in an entry in this way, instead I will use tkinter's StringVar object which changes as the entry's content changes.

```
class LoginScreen(tk.Tk):
    def __init__(self, users, on_login):
        super().__init__()

        self.users = users
        self.on_login = on_login
        self.password = tk.StringVar()
        self.username = tk.StringVar()
```

```
        self.geometry("480x240")
        self.title("GOAT Login")

        ttk.Label(self, text="Username:").pack()
        ttk.Entry(self, textvariable=self.username).pack()

        ttk.Label(self, text="Password:").pack()
        ttk.Entry(self, textvariable=self.password).pack()

        ttk.Button(self, text="Submit", command=self.on_submit).pack()

    def on_submit(self):
        for user in self.users:
            if user.username == self.username.get() and user.password
== self.password.get():
                self.on_login(user)
        self.username.set("")
        self.password.set("")
```

Output:
(After clicking submit)



Test number	Test Description	Test data (if applicable)	Justification	Outcome
17.1	Does the login code correctly show a screen and validate the user's input?	The input Jonathan and "asdfgh" (valid)	The user must be able to login.	As Expected
17.2	Does the login code correctly show a screen and validate the user's input?	The input "aksljdlad" and "ljsdkjfkls" (valid)	The user must not be able to login if they do not enter the correct login details.	As expected
17.3	Do the text entries clear after clicking submit?	n/a	The text entries should clear after clicking submit because if the user is still on the screen after they click submit, it means their username and password was wrong.	Failed and fixed by changing the type of variables used to store the username and password. (**Fig 17.1**)

Login - Prototype 3, A fixed number of attempts and a message to the user

Now I want the user to have 3 attempts at guessing the username and password.

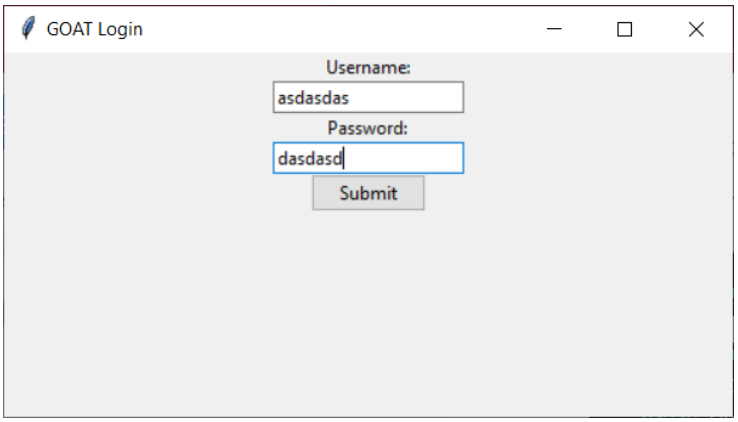
```
class LoginScreen(tk.Tk):
    def __init__(self, users, on_login, attempts_allowed):
        super().__init__()

        self.users = users
        self.on_login = on_login
        self.attempts_remaining = attempts_allowed
        self.password = tk.StringVar()
        self.username = tk.StringVar()

        self.geometry("480x240")
        self.title("GOAT Login")

        ttk.Label(self, text="Username:").pack()
        ttk.Entry(self, textvariable=self.username).pack()

        ttk.Label(self, text="Password:").pack()
        ttk.Entry(self, textvariable=self.password).pack()

        ttk.Button(self, text="Submit", command=self.on_submit).pack()

    def on_submit(self):
        for user in self.users:
            if user.username == self.username.get() and user.password
== self.password.get():
                self.on_login(user)

        self.username.set("")
        self.password.set("")

        self.attempts_remaining -= 1
        if self.attempts_remaining <= 0:
            self.destroy()

        print(self.attempts_remaining)
```

And on the main program

```
def on_login(user):
    print("login successful")
```

```
    print("user is " + str(user))

LoginScreen(users, on_login, attempts_allowed=3).mainloop()
```

Test:



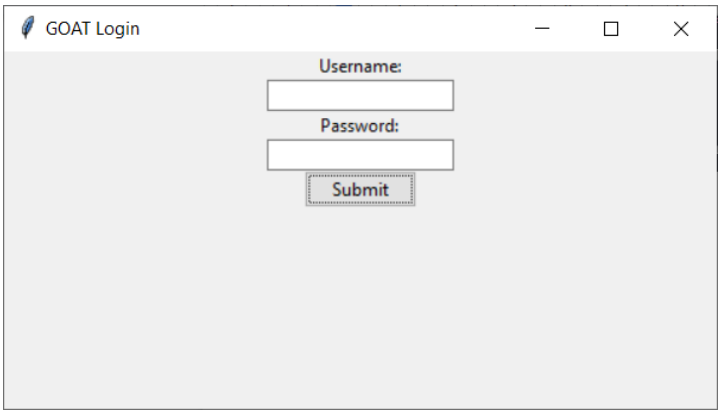
Output:
2
1
0

This works, but the program counts an attempt when there is nothing in the entries (Logic Error)

Fig 18.1
So I'll change the on_submit function to this

```
def on_submit(self):
    # return early if any of the entries are blank
    if self.username.get() == "" or self.password.get() == "":
        return

    for user in self.users:
        if user.username == self.username.get() and user.password ==
self.password.get():
            self.on_login(user)

    self.username.set("")
    self.password.set("")

    self.attempts_remaining -= 1
    if self.attempts_remaining <= 0:
        self.destroy()

    print(self.attempts_remaining)
```

Test:
(Clicking the submit button with empty entries.)



Output:

There is no logging of an attempt in the terminal when the entries are empty.

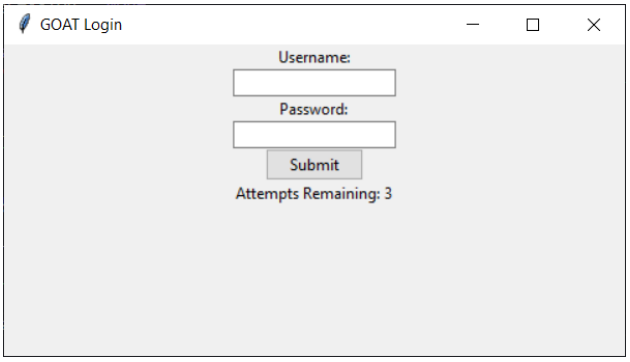
Next, I want the user to be alerted to how many attempts they have remaining and when they haven't filled out one of the required fields, so I add a message attribute to the login screen

```
self.message.set(
    f'Attempts Remaining: {self.attempts_remaining}')
```

And add a label that displays this message

```
# Message to the user
ttk.Label(self, textvariable=self.message).pack()
```

Output:



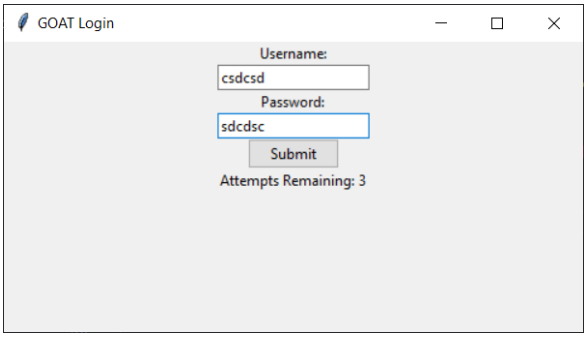
To make this update, I add a method that sets the message to the new number of attempts remaining.

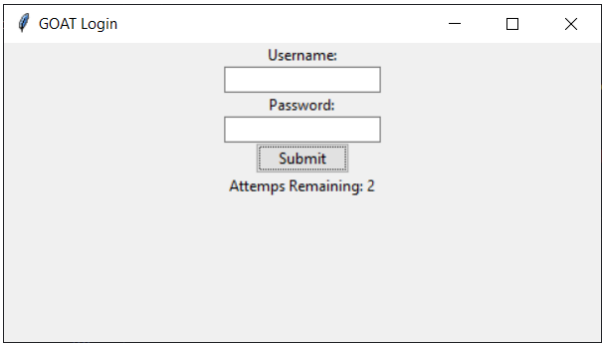
```
def set_message_to_attempts_remaining(self):
    self.message.set(
        f'Attemps Remaining: {self.attempts_remaining}')
```

And then call it after the number of attempts remaining changes.

```
# Update the number of attempts remaining
self.attempts_remaining -= 1
self.set_message_to_attempts_remaining()
if self.attempts_remaining <= 0:
    self.destroy()
```

Output:

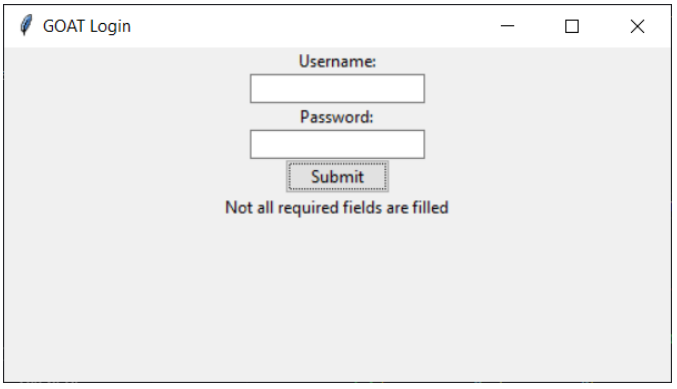




To display the empty field(s) prompt, I add change this part of the on_submit method

```
# return early if any of the entries are blank
if self.username.get() == "" or self.password.get() == "":
    self.message.set("Not all required fields are filled")
    return
```

Output:



I also want the window to disappear after login, so I change the user info checking section of the on_submit method

```
# Check through all of the users to see if any of the login information
matches
for user in self.users:
    if user.username == self.username.get() and user.password ==
self.password.get():
        self.destroy()
        self.on_login(user)
```

Test number	Test Description	Test data (if applicable)	Justification	Outcome
18.1	Can the login code count how many attempts have been taken.	n/a	The number of attempts must be recorded to prevent a brute force attack.	Failed (it counted an attempt when there was text in the entries which is not intended behaviour) and fixed using an early return in the in submit function when the entries are empty. (**Fig 18.1**)
18.2	Can the login screen display a message?	the message (e.g. “Attempt remaining: 3”) (valid)	The user needs to be told how many attempts they have left or what has gone wrong with their login.	As expected
18.3	Does the login screen message update based on the number of attempts remaining?	the number of attempts remaining (3, 2 or 1) (valid)	The user should be told how many attempts they have remaining before they are kicked out of the screen.	As expected
18.4	Does the login screen message update to an appropriate message when the submit button is clicked with empty entries?	n/a	The user should be aware that an empty submit does not count as an attempt and is invalid.	As expected

Account Options screen.

I want new users to be able to create an account and play the game, so there should be a screen before the login screen which gives the user the options to either create an account or login. A screen with different options on it may be useful in the future so I will generalise this idea into an options screen. For an options screen, I need an option class which stores the name of an option (the text on the button) and a function that is called when that option is selected.

```
class Option:
    def __init__(self, name, on_choose):
        self.name = name
        self.on_choose = on_choose
```

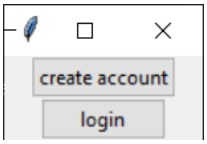
Then the options screen class will take a list of options as an argument of its initialiser. This also uses https://stacktuts.com/tkinter-boiler-plate (August 2022). I will use interaction here because I can have many options on one screen and the process for creating each of the buttons for these options is the same.

```
class OptionsScreen(tk.Tk):
    def __init__(self, options):
        super().__init__()

        for option in options:
            ttk.Button(self, text=option.name,
command=option.on_choose).pack()
```

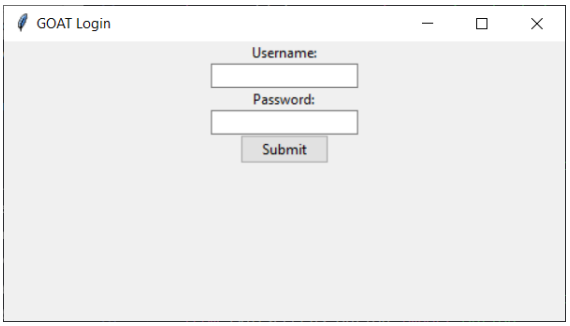
Test:

```
def run_login_screen():
    LoginScreen(users, on_login, attempts_allowed=3).mainloop()

def run_create_account():
    print("create account clicked")

OptionsScreen([
    Option("create account", run_create_account),
    Option("login", run_login_screen)
])
```

Output:



Then after clicking create account
create account clicked
Then after clicking login



But the options screen stays open after an option is clicked (Logic Error).

Fig 19.1

self.destroy should be called before the option function is clicked, so I create a function that adds self.destroy to the start of a function.

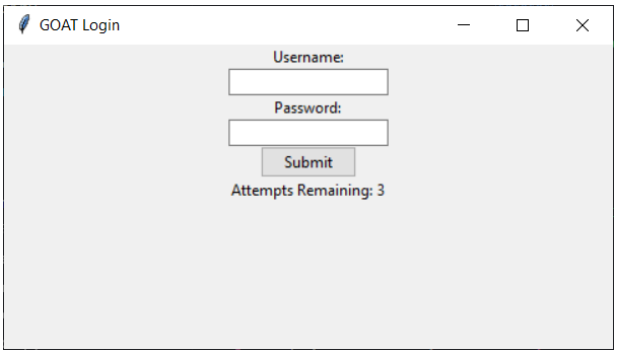
```
# A function that adds self.destroy to the start of a function.
def prepend_destroy(func):
    def result(*args, **kwargs):
        self.destroy()
        func(*args, **kwargs)
    return result
```

Then I change the creation of the buttons to use this function.

```
# Create all the option buttons
for option in options:
    ttk.Button(self, text=option.name,
command=prepend_destroy(option.on_choose)).pack()
```

Output:

(The options screen closes.)



Test number	Test Description	Test data (if applicable)	Justification	Outcome
19.1	Does the account options screen show?	n/a	I need an account options screen so the user can select whether to login or create an account.	As expected
19.2	Does clicking a button on the account options screen run the corresponding function and close the screen properly?	n/a	The account options screen should close after selecting an option to avoid screen clutter.	Failed and fixed by changing the order of the code sequence to destroy the account options screen before running the selected option's corresponding function. (**Fig 19.1**)

Account Creation Screen

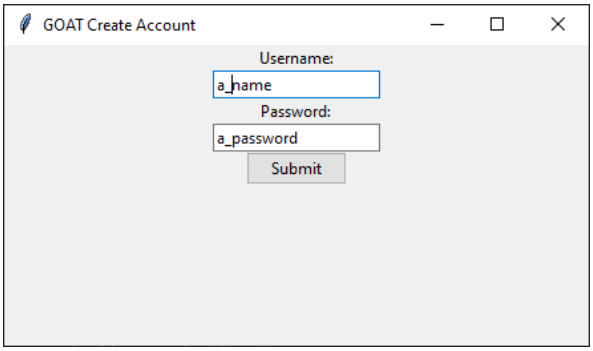
Next, I will create an account creation screen to be used when the user clicks create account, also using https://stacktuts.com/tkinter-boiler-plate (August 2022). Based on the login screen.

```
class CreateAccountScreen(tk.Tk):
    def __init__(self, on_done):
        super().__init__()

        # What happens when the account is created
        self.on_done = on_done
        # Variables that can be used to control the contents of the
screen
        self.username = tk.StringVar()
        self.password = tk.StringVar()
        self.message = tk.StringVar()

        self.geometry("480x240")
        self.title("GOAT Create Account")

        # Username entry
        ttk.Label(self, text="Username:").pack()
        ttk.Entry(self, textvariable=self.username).pack()

        # Password entry
        ttk.Label(self, text="Password:").pack()
        ttk.Entry(self, textvariable=self.password).pack()

        # Submit button
        ttk.Button(self, text="Submit", command=self.on_submit).pack()

        # Any message to the user
        ttk.Label(self, textvariable=self.message).pack()

    def on_submit(self):
        # Get rid of this screen and call the given function
        self.destroy()
        self.on_done(self.username.get(), self.password.get())
```

Test:

```
# logging when the user has logged in
def on_login(user):
    print("login successful")
```

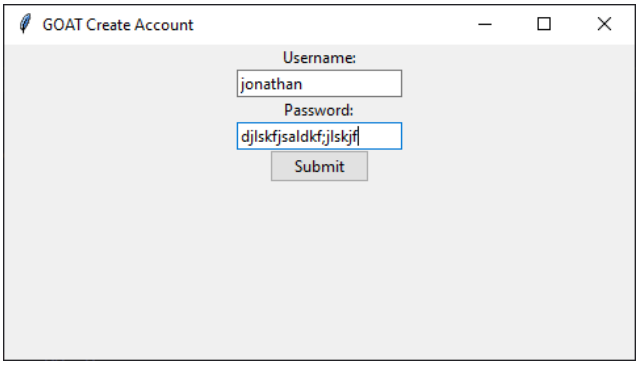
```
    print("user is " + str(user))

# create and run the login screen, using the login logging function and
allowing 3 attempts
def run_login_screen():
    LoginScreen(users, on_login, attempts_allowed=3).mainloop()

# logging when the user has created an account
def create_account(username, password):
    print(f"account created with username {username} and password
{password}")

# Create and run the create account screen using the create account
logging function
def run_create_account_screen():
    CreateAccountScreen(create_account).mainloop()

OptionsScreen([
    Option("create account", run_create_account_screen),
    Option("login", run_login_screen)
]).mainloop()
```



Output:
After clicking submit

account created with username a_name and password a_password

Test:



Output:
(After clicking submit)
account created with username jonathan and password djlskfjsaldkf;jlskjf
This allows the creation of an account with the same username as another account. (Logic Error)

Fig 20.1
To fix this, I will pass the list of users into the account creation screen

```
def __init__(self, users, on_done):
        super().__init__()

        # A list of disallowed usernames
        self.username_blacklist = [user.username for user in users]

        ...
```

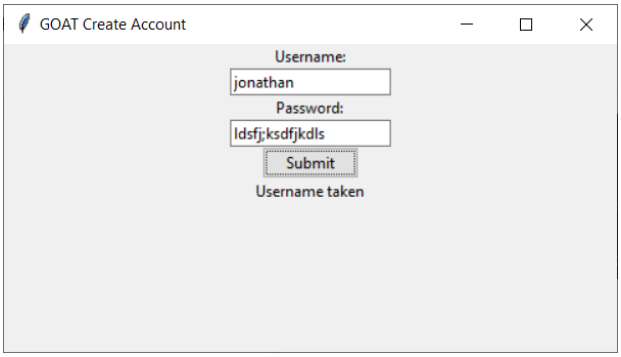
And check to make sure the username does not match with another username on submit

```
def on_submit(self):
    # Disallow submit / account creation when the username entered
already exists
    if self.username.get() in self.username_blacklist:
        self.message.set("Username taken")
        return
    # Get rid of this screen and call the given function
    self.destroy()
    self.on_done(self.username.get(), self.password.get())
```

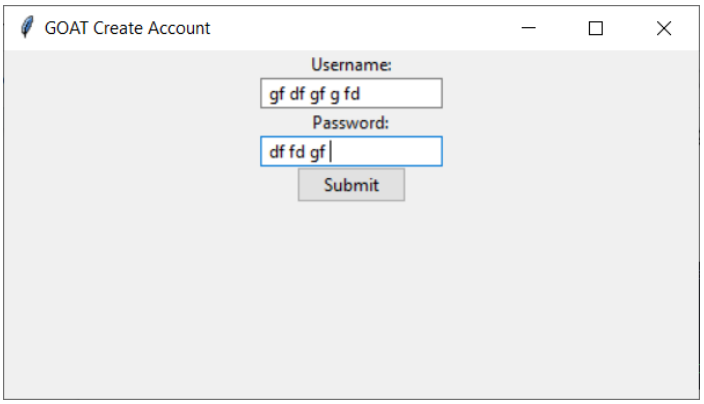
And I'll pass the list of users in

```
# Create and run the create account screen using the create account
logging function
def run_create_account_screen():
    CreateAccountScreen(users, create_account).mainloop()
```

Output:



Test:

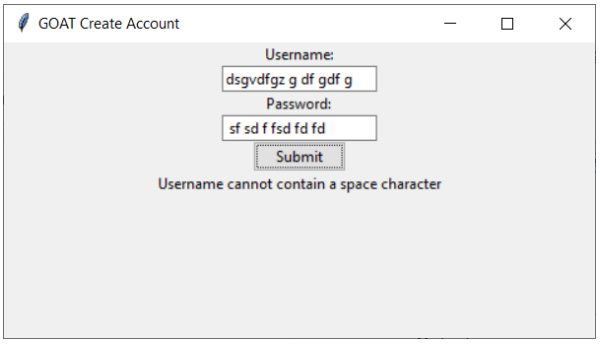


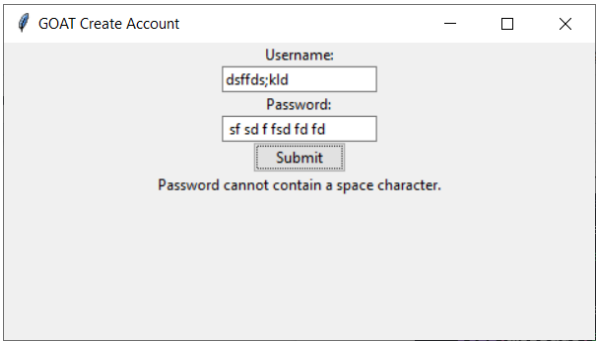
Output:
account created with username gf df gf g fd and password df fd gf (Logic Error)
usernames and passwords with spaces should not be allowed.

Fig 20.2
This disallows account creation and changes the message when a username or password is entered with a space in it.

```
# No spaces allowed
if " " in self.username.get():
    self.message.set("Username cannot contain a space character")
    return
if " " in self.password.get():
    self.message.set("Password cannot contain a space character.")
    return
```

Output:





Next I will add a customisable password filter to the account creation screen, it will be passed into its initialiser.

```
def __init__(self, users, password_filter, on_done):
    super().__init__()

    # A list of disallowed usernames
    self.username_blacklist = [user.username for user in users]

    # A function that only returns true is the password is allowed
```

```
    self.password_filter = password_filter
    ...
```

And used in the on_submit method

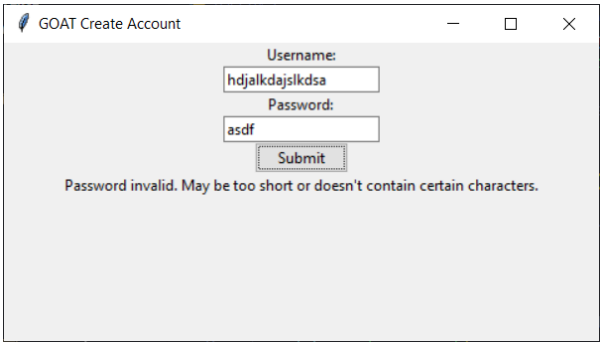
```
def on_submit(self):
    # Disallow submit / account creation when the username entered
already exists
    if self.username.get() in self.username_blacklist:
        self.message.set("Username taken")
        return
    # No spaces allowed
    if " " in self.username.get():
        self.message.set("Username cannot contain a space character")
        return
    if " " in self.password.get():
        self.message.set("Password cannot contain a space character.")
        return
    # Check the password using the password filter
    if not self.password_filter(self.password.get()):
        self.message.set("Password invalid. May be too short or doesn't
contain certain characters.")
        return
    # Get rid of this screen and call the given function
    self.destroy()
    self.on_done(self.username.get(), self.password.get())
```

Then the password filter must be created and passed into the screen.

```
# Create and run the create account screen using the create account
logging function
def run_create_account_screen():
    CreateAccountScreen(users, lambda password : len(password) >= 5,
create_account).mainloop()
```

This should disallow the password if it is shorter than 5 characters.

Test and Output:



Test:

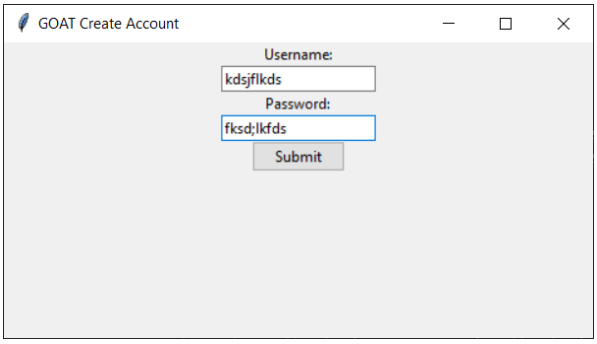
Then when inputting a valid password

Output:
account created with username ndkjflkjdsfk and password amskdlaksdjlaksd

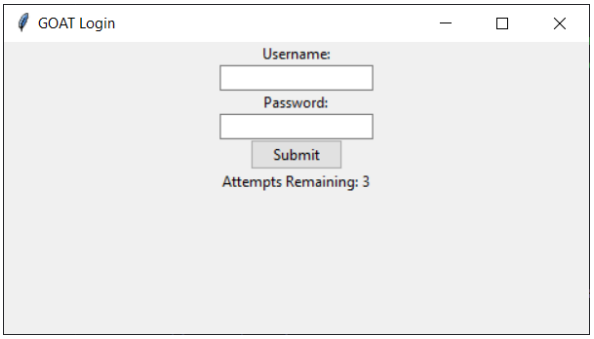
To actually create the account, I need to add the user to the user list and save it to a file. It should also run the login screen straight afterwards.

```
# creating and saving a new user
def create_account(username, password):
    users.append(User(username, password))
    save_obj(users, r"./users.p")
    run_login_screen()
```

Test:



Output:



The new user is saved.

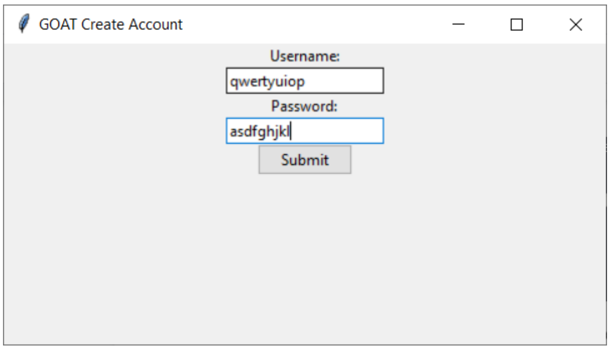
```
►►►►►►►►►►]►(►__main__►►User►►►)►►}►(►username►►jonathan►►password►►asd
fgh►►
progression►Nubh)►►}►(h►bader►h►qwerty►h
Nubh)►►}►(h►  kdsjflkds►h►
fksd;lkfds►h
Nube.
```

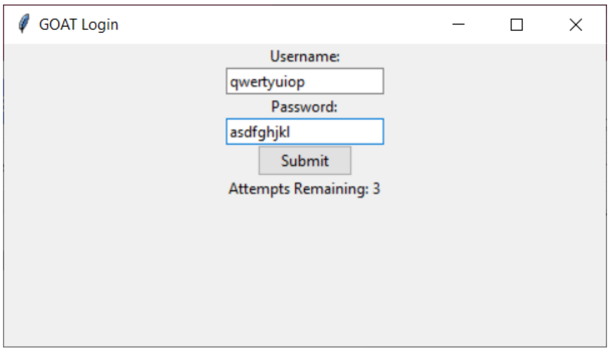
I will now remove the setting users section of code so I can actually save users and they won't be reset each time I run the program.

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
users = [User("jonathan", "asdfgh"), User("bader", "qwerty")]


save_obj(users, r"./users.p")
```

Now I'll add a user and try to login as them.
Test:





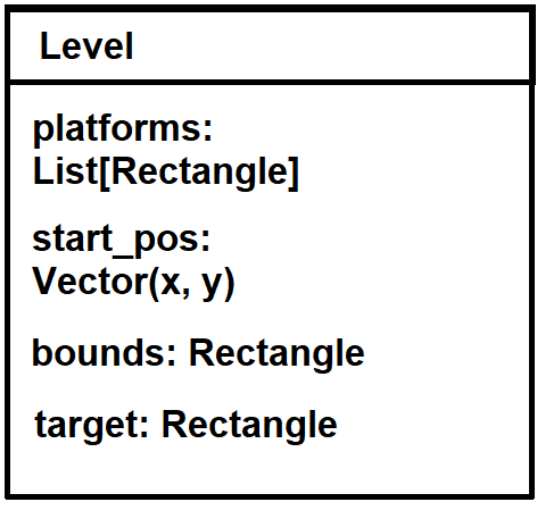
Output:
login successful
user is username: qwertyuiop, password: asdfghjkl, progression: None

Test number	Test Description	Test data (if applicable)	Justification	Outcome
20.1	Can the create account screen successfully take an input and print out the input data?	The data entered into the account creation text entries. (username="a_name", password="a_password") (valid)	I need an account options screen so the user can select whether to login or create an account.	As expected
20.2	Does the create account screen properly reject an input request to create an account with an existing username?	The data entered into the account creation text entries. (username="jonathan", password=<anything>) (valid)	The username is a unique identifier for an account, so duplicate usernames should not be allowed.	Failed and foxed by adding a condition to check whether the username already exists in a list of existing usernames, so as to reject certain inputs. It also gives the user a message saying the username is taken. (**Fig 20.1**)

20.3	Does the create account screen properly reject data with spaces in it?	A username and password with spaces in them e.g. username “gf df gf g fd” and password “df fd gf” (erroneous)	A username and password with spaces may not be stored properly.	Failed and fixed by adding a condition statement which checks the username and password for spaces. (**Fig 20.2**)
20.4	Does the account creation screen reject passwords of less than 5 characters and display a message prompting the user?	A password that is too short. “asdf” (erroneous)	This will ensure that the login system is fairly secure (this is not critical as it is a small game).	As expected
20.5	Does the account creation screen accept a password of 5 characters or more?	An adequate password “amskdlaksdjlaksd” (valid)	Making sure the account creation still works after adding the password security checker.	As Expected
20.6	Can the account creation code actually save a new user to secondary storage?	User info input (valid)	So that the user can login later and continue their progress.	As expected

Levels - Prototype 1, Loading level data from secondary storage

I need to add some way of storing and loading levels. I will use JSON instead of binary (as I did with the user info) so that I can edit the levels more easily as JSON is human-readable. The way I would like to structure the levels is as follows



Where bounds is a rectangle that the player must stay inside (so it doesn't fall off a platform or get somewhere it shouldn't be and keep going forever) and target is a rectangle that the player can collide with to complete the level. I will also add a level_id and a level_number.

The levels will be stored in a levels.json file that will look something like this

```
[
    {
        "level_id": "0",
        "level_number": "1",
        "start_pos": {
            "x": 0,
            "y": 0
        },
        "target": {
            "x": 20,
            "y": 0,
            "hw": 1,
            "hh": 1
        },
        "bounds": {
            "x": 0,
            "y": 0,
            "hw": 100,
```

```
            "hh": 100
        },
        "platforms": [
            {
                "x": 10,
                "y": 2,
                "hw": 50,
                "hh": 0.2
            }
        ]
    }
]
```

To use this, I need to make a level class.

```
class Level:
    def __init__(self, level_id, level_number, start_pos, target,
bounds, platforms):
        self.level_id = level_id
        self.level_number = level_number
        self.start_pos = start_pos
        self.target = target
        self.bounds = bounds
        self.platforms = platforms
```

And to use it in the game, I will change the game code to a function that can take a level.

```
def main(level):
    ...
    game code here
    ...
```

And then call the function like this (the main function won't actually use the level data yet)

```
level = some level
main(level)
```

To load the levels, I will use the json library like this. I will use iteration here because I need to load multiple levels each in turn. The same goes for each platform within these levels.

```
from json import load

# Get a list of levels in the form of dictionaries
with open(r"levels.json", "r") as f:
    level_dicts = load(f)

# Create a list of level objects from the list of level dictionaries
levels = []
for level_dict in level_dicts:
    # Create all the rectangle and vectors
```

```
    start_pos = Vector(level_dict["start_pos"]["x"],
level_dict["start_pos"]["y"])
    # create the target from the level data
    target = Rectangle(
        level_dict["target"]["x"],
        level_dict["target"]["y"],
        level_dict["target"]["hw"],
        level_dict["target"]["hh"],
    )
    # create the bounds from the level data
    bounds = Rectangle(
        level_dict["bounds"]["x"],
        level_dict["bounds"]["y"],
        level_dict["bounds"]["hw"],
        level_dict["bounds"]["hh"],
    )
    # Create the platforms (a list of rectangles)
    platforms = []
    for platform_dict in level_dict["platforms"]:
        platform = Rectangle(
            platform_dict["x"],
            platform_dict["y"],
            platform_dict["hw"],
            platform_dict["hh"],
        )
        platforms.append(platform)
    # Create and add each level to the list
    level = Level(
        level_dict["level_id"],
        level_dict["level_number"],
        start_pos,
        target,
        bounds,
        platforms,
    )
    levels.append(level)
```

Output:

```
  File "C:\Python\Python310\lib\json\__init__.py", line 293, in load
    return loads(fp.read(),
io.UnsupportedOperation: not readable
```

(Logic Error)

Fig 21.1

This happened because I opened the file in write mode, I will change it to read mode.

```
# Get a list of levels in the form of dictionaries
with open(r"levels.json", "r") as f:
    level_dicts = load(f)
```

Output:

```
  File "C:\Python\Python310\lib\json\decoder.py", line 355, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

(Logic Error)

I'll test this by logging the contents of the file.

```
# Get a list of levels in the form of dictionaries
with open(r"levels.json", "r") as f:
    print(f)
    print(f.read())
    level_dicts = load(f)
```

Output:

```
<_io.TextIOWrapper name='levels.json' mode='r' encoding='cp1252'>
```

Fig 21.2

This happens because I'm using the wrong encoding.

```
# Get a list of levels in the form of dictionaries
with open(r"levels.json", "r", encoding="utf-8") as f:
    print(f)
    print(f.read())
    level_dicts = load(f)
```

Output:

```
<_io.TextIOWrapper name='levels.json' mode='r' encoding='utf-8'>
```

The levels.json file didn't register as proper JSON because it was empty as I had been using write mode which overwrote the entire file.

Output:

```
<_io.TextIOWrapper name='levels.json' mode='r'
encoding='utf-8'>
[
   {
      "level_id": "0",
      "level_number": "1",
      "start_pos": {
         "x": 0,
         "y": 0
```

```
        },
        "target": {
            "x": 20,
            "y": 0,
            "hw": 1,
            "hh": 1
        },
        "bounds": {
            "x": 0,
            "y": 0,
            "hw": 100,
            "hh": 100
        },
        "platforms": [
            {
                "x": 10,
                "y": 2,
                "hw": 50,
                "hh": 0.2
            }
        ]
    }
]
  File "C:\Python\Python310\lib\json\decoder.py", line 355, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value:
line 1 column 1 (char 0)
```

(Logic Error)

Fig 21.3

json.load couldn't read the file because it has already been read.

```
# Get a list of levels in the form of dictionaries
with open(r"levels.json", "r", encoding="utf-8") as f:
    level_dicts = load(f)
```

Output:

```
  File "D:\usb for win down backup (goat)\GOATwvenv\game\game.py", line 26, in main
    player = Player(level.start_pos.x, level.start_pos.y, 0.5, 0.5, vel_x=0, vel_y=1)
AttributeError: 'str' object has no attribute 'start_pos'
```

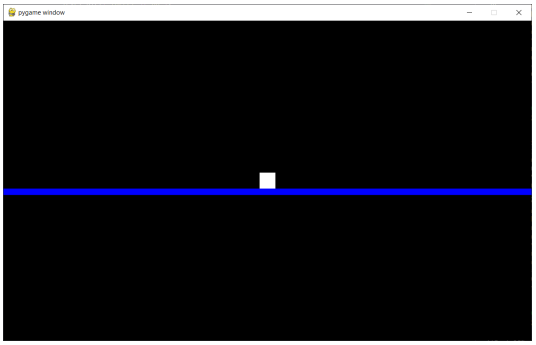
(Logic Error)

Fig 21.4

I hadn't changed the level actually being used in the main function from the placeholder to a level in the list.

```
level = levels[0]
main(level)
```

Output:



Test number	Test Description	Test data (if applicable)	Justification	Outcome
21.1	Is the level data loaded correctly?	Data in the levels.json file (valid)	The level data must be loaded from secondary storage where it is stored to main memory where it will be processed and used.	Failed and fixed by: Switching from write mode to read mode (**Fig 21.1**). Changing the encoding used (**Fig 21.2**). Removing unnecessary reading for the file (**Fig 21.3**). Switching the level used to the first one in the list in the json file (**Fig 21.4**).

Levels - Prototype 2, Using loaded level data in the gameplay

Now I will turn the game code into a function that takes a screen and draws on it like the menus do, using the data in the loaded level.

```
def game(screen, level):

    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    player = Player(level.start_pos.x, level.start_pos.y, 0.5, 0.5,
vel_x=0, vel_y=1)
    camera = Camera(0, 5, WIDTH, HEIGHT, game_space_visible_height=20)

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return True

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        player.reset_acc()

        # acceleration
        player.gravity(100)
        player.move_horizontal(
```

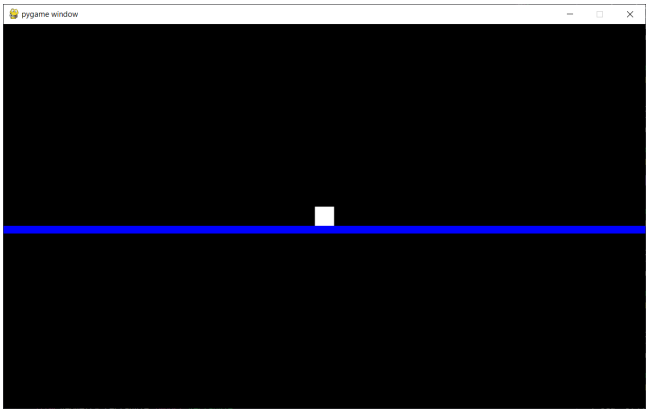
```
            left_input=keys[pygame.K_a],
            right_input=keys[pygame.K_d],
            grounded_speed=150,
            ungrounded_speed=10,
        )
        player.jump(jump_input=keys[pygame.K_SPACE], power=2000)
        # player.air_resistance(power=0.5)
        player.ground_friction(power=10)

        # update displacement and velocity
        player.update_displacement_and_velocity(dt)

        player.reset_grounded()

        # collisions
        for p in level.platforms:
            player.collide_with_rectangle(p)

        # update position
        player.update_position()

        camera.pos = player.rect.pos

        pygame.draw.rect(
            screen,
            WHITE,
            camera.game_space_to_screen_space_corner_rect_tuple(
                player.rect.corner_rect_tuple()
            ),
        )

        for p in level.platforms:
            pygame.draw.rect(
                screen,
                BLUE,
                camera.game_space_to_screen_space_corner_rect_tuple(
                    p.corner_rect_tuple()
                ),
            )

        pygame.display.flip()
```

Test:

```
WIDTH = 1000
HEIGHT = 600

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.font.init()

from level import Level
from geometry.rectangle import Rectangle

from json import load

# Get a list of levels in the form of dictionaries
with open(r"levels.json", "r", encoding="utf-8") as f:
    level_dicts = load(f)

# Create a list of level objects from the list of level dictionaries
levels = []
for level_dict in level_dicts:
    # Create all the rectangle and vectors
    start_pos = Vector(level_dict["start_pos"]["x"],
level_dict["start_pos"]["y"])
    target = Rectangle(
        level_dict["target"]["x"],
        level_dict["target"]["y"],
        level_dict["target"]["hw"],
        level_dict["target"]["hh"],
    )
    bounds = Rectangle(
        level_dict["bounds"]["x"],
        level_dict["bounds"]["y"],
        level_dict["bounds"]["hw"],
        level_dict["bounds"]["hh"],
    )
    # Create the platforms (a list of rectangles)
    platforms = []
    for platform_dict in level_dict["platforms"]:
        platform = Rectangle(
            platform_dict["x"],
            platform_dict["y"],
            platform_dict["hw"],
            platform_dict["hh"],
```

```
        )
        platforms.append(platform)
    # Create and add each level to the list
    level = Level(
        level_dict["level_id"],
        level_dict["level_number"],
        start_pos,
        target,
        bounds,
        platforms,
    )
    levels.append(level)

level = levels[0]

game(screen, level)

# this will only run when all the loop in the root screen has finished
pygame.quit()
```

Output:



Test number	Test Description	Test data (if applicable)	Justification	Outcome
22.1	Is the level data that has been loaded from secondary storage processed and displayed properly by the gameplay function.	Data in the level object. (valid)	The level data must be properly processed so that all the levels can be used by the game and provide adequate content for the user.	As expected

Levels - Prototype 3, the level select screen

Now, I will change the level_select screen (previously just used as an example to test the menu system) to display all the levels and allow the user to select one of them.

```
import pygame

from geometry.vector import Vector

from .button import Button

from level import Level
from geometry.rectangle import Rectangle
from game.game import game

from json import load


def load_levels(levels_file_path):
    # Get a list of levels in the form of dictionaries
    with open(levels_file_path, "r", encoding="utf-8") as f:
        level_dicts = load(f)

    # Create a list of level objects from the list of level
dictionaries
    levels = []
    for level_dict in level_dicts:
        # Create all the rectangle and vectors
        start_pos = Vector(level_dict["start_pos"]["x"],
level_dict["start_pos"]["y"])
        target = Rectangle(
            level_dict["target"]["x"],
            level_dict["target"]["y"],
            level_dict["target"]["hw"],
            level_dict["target"]["hh"],
        )
        bounds = Rectangle(
            level_dict["bounds"]["x"],
            level_dict["bounds"]["y"],
            level_dict["bounds"]["hw"],
            level_dict["bounds"]["hh"],
        )
        # Create the platforms (a list of rectangles)
        platforms = []
```

```
        for platform_dict in level_dict["platforms"]:
            platform = Rectangle(
                platform_dict["x"],
                platform_dict["y"],
                platform_dict["hw"],
                platform_dict["hh"],
            )
            platforms.append(platform)
        # Create and add each level to the list
        level = Level(
            level_dict["level_id"],
            level_dict["level_number"],
            start_pos,
            target,
            bounds,
            platforms,
        )
        levels.append(level)

    return levels


def level_select(screen):
    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    # create some text to show which screen it is
    my_font = pygame.font.SysFont("Comic Sans MS", 50)

    # load all the levels and create buttons from them
    level_buttons = [
        Button(
            f"Level {level.level_number}, id: {level.level_id}",
```

```
            i * 100 + 200,
            200,
            40,
            40,
            lambda : game(screen, level),
            my_font,
            WHITE
        )
        for i, level in enumerate(load_levels(r"levels.json"))
    ]

    text_render = my_font.render("level_select", False, WHITE)

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        mouse_pos = Vector.from_tuple(pygame.mouse.get_pos())

        for event in pygame.event.get():
            # go all the way back if cross clicked
            if event.type == pygame.QUIT:
                return True
            # check all buttons for mouse down
            if event.type == pygame.MOUSEBUTTONDOWN:
                for level_button in level_buttons:
                    level_button.check_mouse_down(mouse_pos)

            # check all buttons for mouse up
            if event.type == pygame.MOUSEBUTTONUP:
                for level_button in level_buttons:
                    # this will run the game code if the button is
clicked
                    # and pass down the exit command if the cross is
clicked
                    if level_button.check_mouse_up(mouse_pos):
                        return True

            if event.type == pygame.KEYDOWN:
                # go back if Esc clicked
                if event.key == pygame.K_ESCAPE:
                    return
```

```
        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        screen.blit(text_render, (0, 0))

        pygame.display.flip()
```

Test:

```
WIDTH = 1000
HEIGHT = 600

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.font.init()

main_menu(screen)

# this will only run when all the loop in the root screen has finished
pygame.quit()
```

Output:

(Logic Error)

Fig 23.1

I didn't add the code to draw the buttons, so they aren't shown on the screen.

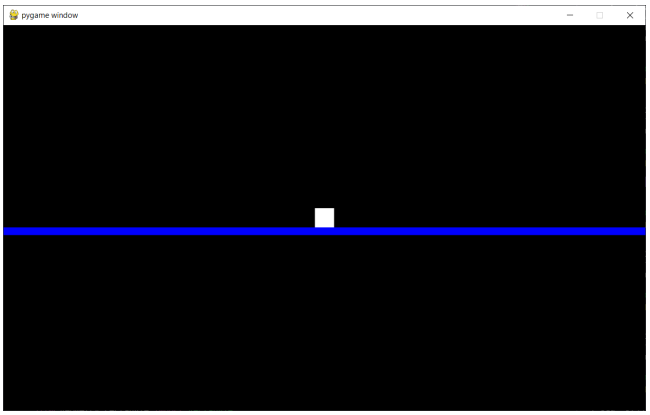
```
def level_select(screen):

    ...

    running = True
    while running:
```

```
        ...

        screen.fill(BLACK)

        screen.blit(text_render, (0, 0))

        for level_button in level_buttons:
            level_button.draw(screen)

        pygame.display.flip()
```

Output:

Then after clicking level_select

Then after clicking Level 1



Test number	Test Description	Test data (if applicable)	Justification	Outcome
23.1	Does the level select button take the user to the level select screen?	n/a	The user must be able to navigate to the level select screen to be able to select a level?	As expected
23.2	Is a level selection button shown on the level select screen?	n/a	The user must be able to see the level select buttons to select one.	Failed and fixed by adding code in the game loop to draw the buttons (**Fig 23.1**)
23.3	Does clicking on a level button take the user to the associated level?	n/a	The level select buttons must take the user to a level so that the user can access the levels.	As Expected

Levels - Prototype 4, Implementing the target

Now, I'll change the game code to make the target (which I previously added to the levels.json file and level class) work.
I'll continually check for collisions with the target's hitbox and print out a message if this occurs (which means the player has completed the level).
I changed the collision method to return true if there is a collision and false if not.
And I added this to the game loop

```
if player.collide_with_rectangle(target):
            print("level completed")


...


pygame.draw.rect(
            screen,
            GREEN,
            camera.game_space_to_screen_space_corner_rect_tuple(
                level.target.corner_rect_tuple()
            )
        )
```

Output:

```
  File "D:\usb for win down backup (goat)\GOATwvenv\game\game.py", line 68, in game
    if player.collide_with_rectangle(target):
NameError: name 'target' is not defined
```

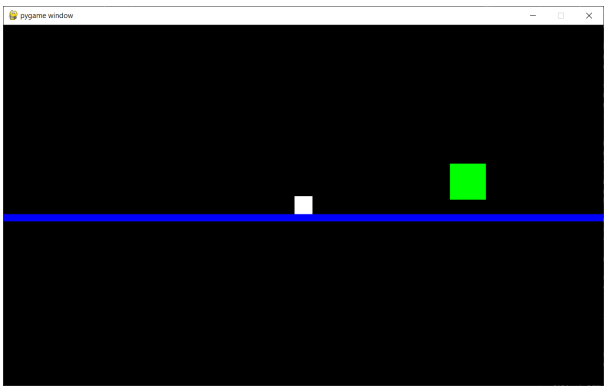
(Logic Error)

Fig 24.1

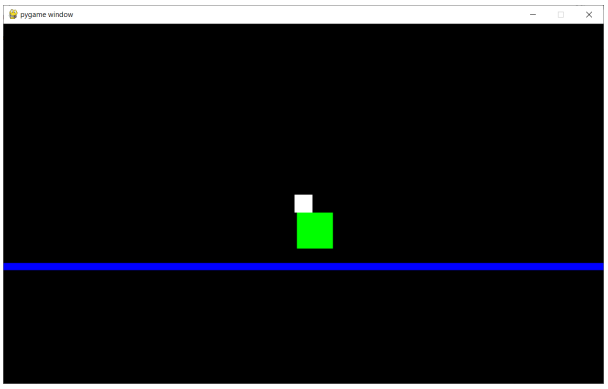
I fixed this by changing the collision code to this

```
if player.collide_with_rectangle(level.target):
    print("level completed")
```

Output:



level completed
level completed
level completed
level completed
level completed
level completed
level completed
level completed
level completed
level completed



level completed

But “level completed” is only printed sometimes. (Logic Error)

Fig 24.2

This is because I placed the

```
return True
```

statement too far indented.

So I fixed that.

Output:
“level completed” is printed all the time the player is in contact with the green block.

To tidy the code up a bit, the game function is turned into a function that can be in the menu stack like all the other menu screens (the content of the function doesn’t change, this is just some reorganisation).

I don’t really want the player to collide with the target, so I’ll split the player’s collision code into two methods for detection and response.

```
def rectangle_collision_detection(self, rect):
    """
    two rectangles colliding can be simplified to a rectangle and point
colliding
    """
```

```
    # combined rectangle
    r = rect.combine_rectangle(self.rect)
    # point
    p = self.rect.pos
    # vector
    v = self.dis
    # end point
    e = p + v

    # positions of the sides of the rectangle
    right = r.right_pos()
    bottom = r.bottom_pos()
    left = r.left_pos()
    top = r.top_pos()

    if (
        (p.x >= right and e.x >= right)
        or (p.y >= bottom and e.y >= bottom)
        or (p.x <= left and e.x <= left)
        or (p.y <= top and e.y <= top)
    ):
        return False

    # vectors from p to each corner of r
    br_cv = r.bottom_right_pos() - p
    bl_cv = r.bottom_left_pos() - p
    tr_cv = r.top_right_pos() - p
    tl_cv = r.top_left_pos() - p

    # sign (+/-) of dot product of corner vector with v
    dp_br_sign = Vector.dot(v, br_cv) > 0.0
    dp_bl_sign = Vector.dot(v, bl_cv) > 0.0
    dp_tl_sign = Vector.dot(v, tl_cv) > 0.0
    dp_tr_sign = Vector.dot(v, tr_cv) > 0.0

    # this may be covered by the first return case
    if (
        (not dp_br_sign)
        and (not dp_bl_sign)
        and (not dp_tl_sign)
        and (not dp_tr_sign)
    ):
        return False
```

```
    # v rotated 90 degrees
    perp_v = v.rotate90()

    # sign (+/-) of dot product of corner vector with v rotated 90
degrees
    perp_dp_br_sign = Vector.dot(perp_v, br_cv) > 0.0
    perp_dp_bl_sign = Vector.dot(perp_v, bl_cv) > 0.0
    perp_dp_tl_sign = Vector.dot(perp_v, tl_cv) > 0.0
    perp_dp_tr_sign = Vector.dot(perp_v, tr_cv) > 0.0

    if (
        perp_dp_br_sign and perp_dp_bl_sign and perp_dp_tl_sign and
perp_dp_tr_sign
    ) or (
        (not perp_dp_br_sign)
        and (not perp_dp_bl_sign)
        and (not perp_dp_tl_sign)
        and (not perp_dp_tr_sign)
    ):
        return False

    return {
        "v": v,
        "right": right,
        "bottom": bottom,
        "left": left,
        "top": top,
        "perp_dp_br_sign": perp_dp_br_sign,
        "perp_dp_bl_sign": perp_dp_bl_sign,
        "perp_dp_tl_sign": perp_dp_tl_sign,
        "perp_dp_tr_sign": perp_dp_tr_sign,
    }

def rectangle_collision_response(self, collision):
    v = collision["v"]
    right = collision["right"]
    bottom = collision["bottom"]
    left = collision["left"]
    top = collision["top"]
    perp_dp_br_sign = collision["perp_dp_br_sign"]
    perp_dp_bl_sign = collision["perp_dp_bl_sign"]
    perp_dp_tl_sign = collision["perp_dp_tl_sign"]
```

```
    perp_dp_tr_sign = collision["perp_dp_tr_sign"]
    if v.x > 0.0:
        if v.y > 0.0:
            if perp_dp_tl_sign:
                # top
                self.no_bounce_collision_y(top)
                self.set_grounded()
            else:
                # left
                self.no_bounce_collision_x(left)
        else:
            if perp_dp_bl_sign:
                # left
                self.no_bounce_collision_x(left)
            else:
                # bottom
                self.no_bounce_collision_y(bottom)
    else:
        if v.y > 0.0:
            if perp_dp_tr_sign:
                # right
                self.no_bounce_collision_x(right)
            else:
                # top
                self.no_bounce_collision_y(top)
                self.set_grounded()
        else:
            if perp_dp_br_sign:
                # bottom
                self.no_bounce_collision_y(bottom)
            else:
                # right
                self.no_bounce_collision_x(right)

# continuous collision detection
def collide_with_rectangle(self, rect):
    collision = self.rectangle_collision_detection(rect)
    self.rectangle_collision_response(collision)
```

And I changed the target collision code to this

```
if player.rectangle_collision_detection(level.target):
    print("level completed")
```

Output:

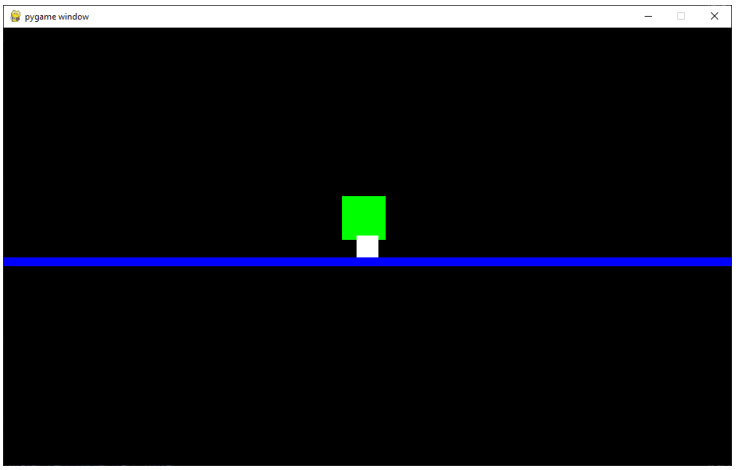
```
  File "D:\usb for win down backup (goat)\GOATwvenv\game\player.py",
line 231, in rectangle_collision_response
    v = collision["v"]
TypeError: 'bool' object is not subscriptable
```

(Logic Error)

This happens because collision response code is being called even when there is no collision, so the collision response code tries to get information from a false value. To fix this, I will make the response method return early if there is no collision.

```
def rectangle_collision_response(self, collision):
    if not collision:
        return

    ...
```

Output:



level completed
level completed
level completed
level completed
level completed
level completed
level completed
level completed
level completed
level completed
level completed

Test number	Test Description	Test data (if applicable)	Justification	Outcome
24.1	Does the target show up in a level?	the position of the target (10, -1) (valid)	The user must be able to see the target to know how to complete the level. This also shows that the game code can properly process the target.	Failed and fixed changing the variable used to access the target to an attribute on the level object (**Fig 24.1**)
24.2	Are collisions detected properly between the player and the target?	n/a	The collision detection between the player and the target must work all the time for the user to be able to complete the level.	Failed and fixed by changing where the return statement is in the collision code (**Fig 24.2**)
24.3	Does the new method of splitting the collision code into separate functions for collision detection and response work?	n/a	This will make my code more modular and therefore reusable.	Failed and fixed by adding an early return condition to the collision response code so that the program doesn't try to process a null collision (**Fig 24.3**).

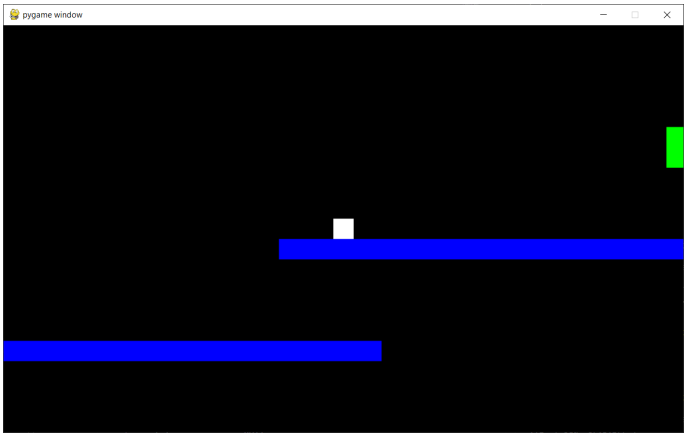
Levels - Prototype 5, multiple levels

I would now like to add some more test levels.
So I add this to levels.json

```
{
        "level_id": "1",
        "level_number": "2",
        "start_pos": {
            "x": 0,
            "y": -10
        },
        "target": {
            "x": 25,
            "y": -15,
            "hw": 1,
            "hh": 1
        },
        "bounds": {
            "x": 0,
            "y": 0,
            "hw": 100,
            "hh": 100
        },
        "platforms": [
            {
                "x": 0,
                "y": -5,
                "hw": 10,
                "hh": 0.5
            },
            {
                "x": 15,
                "y": -10,
                "hw": 10,
                "hh": 0.5
            }
        ]
    }
```

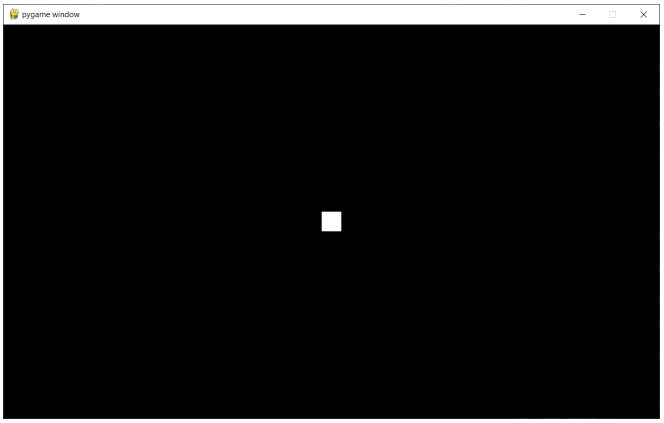
Output:
When viewing the levels screen

When clicking on the level 2 button



When jumping to the higher platform, I notice that sometimes I can jump high enough, and sometimes I can't. (Logic Error)
I must fix this, or the gameplay will be inconsistent.
I think this must be something to do with how acceleration is applied at different frame times.
To test this I will change the frame rate of the game to 10 frames per second.

Output:
The player moves the same horizontally.
Jumping looks like this



The reason this happens is because when the player jump, an upwards acceleration is applied for the duration of the frame, so the longer the frame is, the longer the upwards acceleration is applied for, so when the frame rate is lower, the acceleration is applied for a longer period of time, so the upwards velocity at the start of the jump is greater. This also means that if I set the frame rate to 120 frames per second, the jump is a lot smaller.

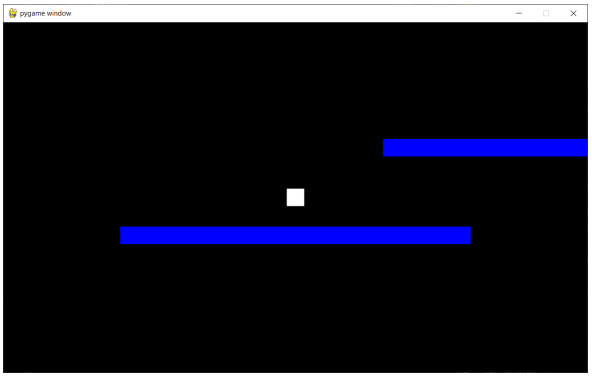
So I change the jump method on the player to respond to the time between frames.

```
def jump(self, jump_input, power, dt):
    if jump_input and self.grounded:
        self.apply_acc(Vector(0, -power / dt))
```

And I change the call of the method to this (also adjusting the power argument)

```
player.jump(jump_input=keys[pygame.K_SPACE], power=35, dt=dt)
```

Output:



The jump height is normal, regardless of framerate.

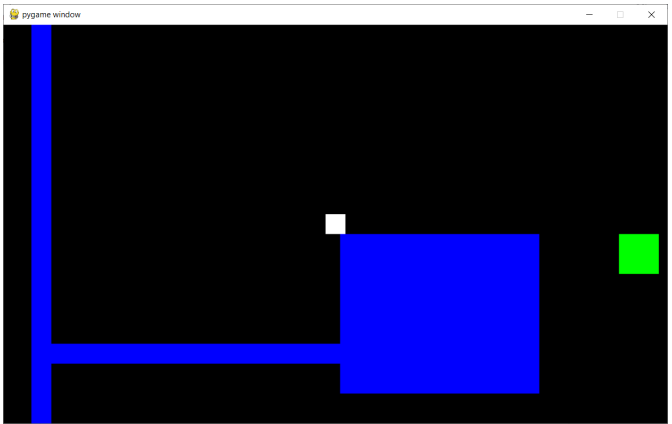
I will now add another level for testing by adding this to levels.json

```
{
    "level_id": "2",
    "level_number": "3",
    "start_pos": {
        "x": 0,
        "y": -10
    },
    "target": {
        "x": 20,
        "y": -5,
        "hw": 1,
        "hh": 1
    },
    "bounds": {
        "x": 0,
        "y": 0,
        "hw": 100,
```

```
        "hh": 100
    },
    "platforms": [
        {
            "x": -10,
            "y": 0,
            "hw": 0.5,
            "hh": 20
        },
        {
            "x": 0,
            "y": 0,
            "hw": 10,
            "hh": 0.5
        },
        {
            "x": 10,
            "y": -2,
            "hw": 5,
            "hh": 4
        }
    ]
}
```

Output:
On level select screen

After clicking the level 3 button



But after clicking the level 2 button, level 3 is loaded. (Logic Error)
This is a bug that I missed when I created the level selection screen.
It happens because the level being level being used in the gameplay function here

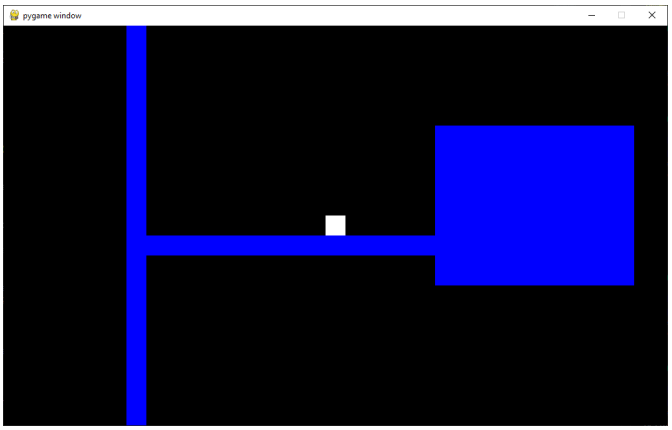
```
lambda: gameplay(screen, level)
```

changes during the creation of the level buttons, so all the level buttons end up directing the use to the last level in the list.

Fig 25.1

This can be fixed by hard coding level functions like this

```
level_functions = [
        lambda: gameplay(screen, levels[0]),
        lambda: gameplay(screen, levels[1]),
        lambda: gameplay(screen, levels[2]),
    ]
```



But this means that this piece of code has to be changed every time a new level is added.

Fig 25.2

After a lot of testing, I found a way to solve this problem. I have to create a wrapper function like this

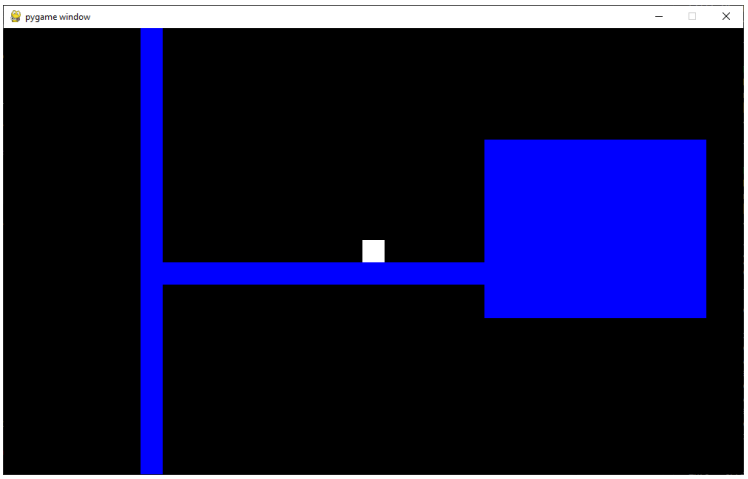
```
def create_level_function(level):
    def level_function():
        return gameplay(screen, level)
    return level_function
```

This preserves the state of the level in the wrapper function so the value of level doesn't change as the other buttons are being created.
This turns the level buttons creation code into this

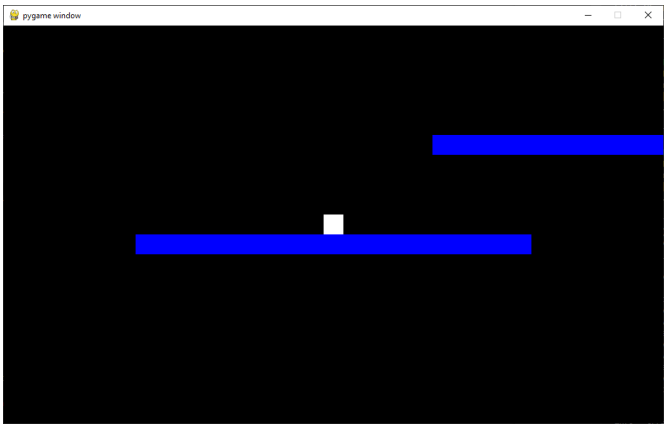
```
def create_level_function(level):
    def level_function():
        return gameplay(screen, level)
    return level_function


level_buttons = [
    Button(
```

```
        f"Level {levels[i].level_number}, id: {levels[i].level_id}",
        i * 100 + 200,
        200,
        40,
        40,
        create_level_function(level),
        my_font,
        WHITE,
    )
    for i, level in enumerate(levels)
]
```

Output:
After clicking the level 3 button



After clicking the level 2 button



The code could be shortened to this

```
level_buttons = [
    Button(
        f"Level {levels[i].level_number}, id: {levels[i].level_id}",
        i * 100 + 200,
        200,
        40,
        40,
        (lambda level: lambda: gameplay(screen, level))(level),
        my_font,
        WHITE,
    )
    for i, level in enumerate(levels)
]
```

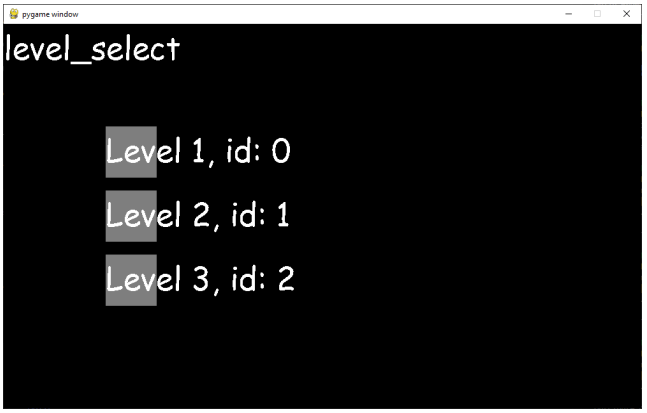
but this creates a new level function creator function for each level, which is not efficient. Also, it is quite difficult to understand what it does, whereas this

```
def create_level_function(level):
    def level_function():
```

```
        return gameplay(screen, level)
    return level_function
```

is very clearly a wrapper function, so I'll stick with the more verbose option.

I will change the level selection screen to make it a bit more clear.



Test number	Test Description	Test data (if applicable)	Justification	Outcome
25.1	Does the level 2 button appear and load level 2 when clicked?	n/a	I must have multiple levels to keep the user engaged.	As expected
25.2	Does the level 3 button appear and load level 3 when clicked?	n/a	I must have multiple levels to keep the user engaged.	As Expected
25.3	Do both level buttons work when they are both together?	n/a	I must have many levels, so more than just two must work.	Failed and fixed by having separate hardcoded functions for the level buttons (**Fig 25.1**). And then using a wrapper function (**Fig 25.2**).

Levels - Prototype 6, level ordering

I have changed my mind. Each level in the levels.json file should have an id, but not a level number. The JSON file should just be an unordered collection of levels. The level order should be determined by a list of ids. This way, the order of levels can be changed and inserting a level in the middle of all the other levels will be easier.

So now levels.json looks like this

```
[
    {
        "level_id": "0",
        "start_pos": {
            "x": 0,
            "y": 0
        },
        "target": {
            "x": 20,
            "y": 0,
            "hw": 1,
            "hh": 1
        },
        "bounds": {
            "x": 0,
            "y": 0,
            "hw": 100,
            "hh": 100
        },
        "platforms": [
            {
                "x": 10,
                "y": 2,
                "hw": 50,
                "hh": 0.2
            }
        ]
    },
    {
        "level_id": "1",
        "start_pos": {
            "x": 0,
            "y": -10
        },
        "target": {
```

```
            "x": 25,
            "y": -15,
            "hw": 1,
            "hh": 1
        },
        "bounds": {
            "x": 0,
            "y": 0,
            "hw": 100,
            "hh": 100
        },
        "platforms": [
            {
                "x": 0,
                "y": -5,
                "hw": 10,
                "hh": 0.5
            },
            {
                "x": 15,
                "y": -10,
                "hw": 10,
                "hh": 0.5
            }
        ]
    },
    {
        "level_id": "2",
        "start_pos": {
            "x": 0,
            "y": -10
        },
        "target": {
            "x": 20,
            "y": -5,
            "hw": 1,
            "hh": 1
        },
        "bounds": {
            "x": 0,
            "y": 0,
            "hw": 100,
            "hh": 100
```

```
        },
        "platforms": [
            {
                "x": -10,
                "y": 0,
                "hw": 0.5,
                "hh": 20
            },
            {
                "x": 0,
                "y": 0,
                "hw": 10,
                "hh": 0.5
            },
            {
                "x": 10,
                "y": -2,
                "hw": 5,
                "hh": 4
            }
        ]
    }
]
```

I also update the level class to match this

And I'll create a new JSON file containing the order of the levels.

```
[
    "0",
    "1",
    "2"
]
```

And I change the load_levels function to take a path to this file and use the contents of this file to order the levels.

```
def load_levels(levels_file_path, levels_order_file_path):
    # Get a list of levels in the form of dictionaries
    with open(levels_file_path, "r", encoding="utf-8") as f:
        level_dicts = load(f)

    # Create a list of level objects from the list of level
dictionaries
    levels = []
    for level_dict in level_dicts:
        # Create all the rectangle and vectors
```

```
        start_pos = Vector(level_dict["start_pos"]["x"],
level_dict["start_pos"]["y"])
        target = Rectangle(
            level_dict["target"]["x"],
            level_dict["target"]["y"],
            level_dict["target"]["hw"],
            level_dict["target"]["hh"],
        )
        bounds = Rectangle(
            level_dict["bounds"]["x"],
            level_dict["bounds"]["y"],
            level_dict["bounds"]["hw"],
            level_dict["bounds"]["hh"],
        )
        # Create the platforms (a list of rectangles)
        platforms = []
        for platform_dict in level_dict["platforms"]:
            platform = Rectangle(
                platform_dict["x"],
                platform_dict["y"],
                platform_dict["hw"],
                platform_dict["hh"],
            )
            platforms.append(platform)
        # Create and add each level to the list
        level = Level(
            level_dict["level_id"],
            start_pos,
            target,
            bounds,
            platforms,
        )
        levels.append(level)

    with open(levels_order_file_path, "r", encoding="utf-8") as f:
        levels_order = load(f)

    ordered_levels = []
    for level_id in levels_order:
        for level in levels:
            if level.level_id == level_id:
                ordered_levels.append(level)
                break
```

```
    return ordered_levels
```

And then I update where this function is called in the level select screen code.

```
# load levels from the json file
levels = load_levels(r"levels/levels.json", r"levels/levelsorder.json")
```

Output:

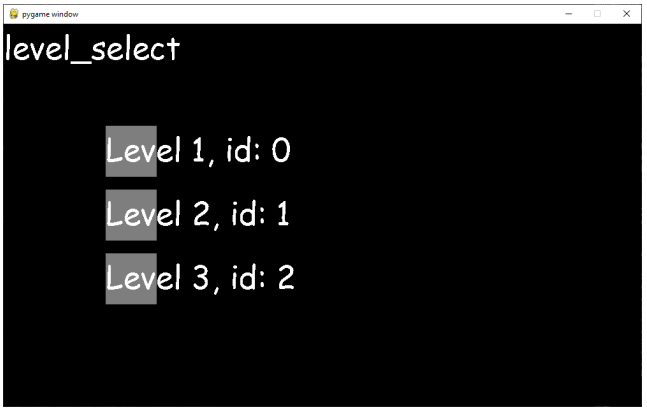
File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\levelselect.py", line 42, in <listcomp>
f"Level {levels[i].level_number}, id: {levels[i].level_id}",
AttributeError: 'Level' object has no attribute 'level_number'
(Logic Error)

Fig 26.1

This happens because I haven't updated the button creation code.

```
# create the level buttons
level_buttons = [
    Button(
        f"Level {i + 1}, id: {levels[i].level_id}",
        200,
        200 + 100 * i,
        40,
        40,
        create_level_function(level),
        # (lambda level: lambda: gameplay(screen, level))(level),
        my_font,
        WHITE,
    )
    for i, level in enumerate(levels)
]
```

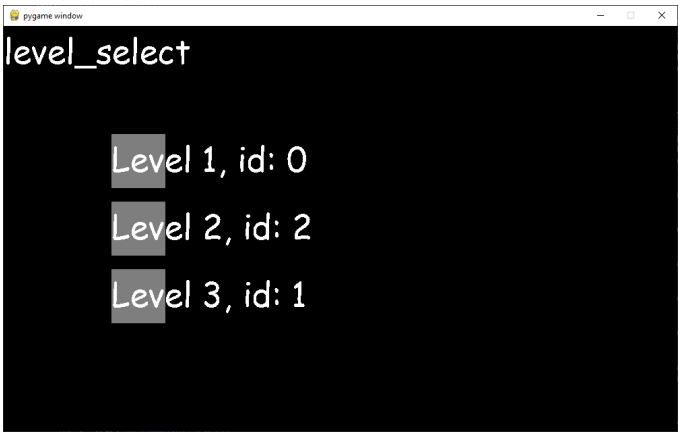
Output:



Then after changing the "levelsorder.json" file to this

```
[
```

```
    "0",
    "2",
    "1"
]
```



Test number	Test Description	Test data (if applicable)	Justification	Outcome
26.1	Does the level ordering system using the levels order JSON file work?	The data in the levels order JSON file [“0”, “1”, “2”] (valid)	The levels must be ordered correctly so there can be some sensible difficulty progression.	Failed and fixed by changing the code used to create the level select buttons (**Fig 26.1**).

Pause Screen

When the user presses esc on the gameplay screen, I want a pause menu to appear instead of being taken back to the level select screen.

To do this, I will create a pause screen function that has some buttons for resuming the game and exiting the game. It will be similar to the main menu.

```
def pause_menu(screen):

    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    my_font = pygame.font.SysFont("Comic Sans MS", 50)

    resume_button = Button(
        "resume",
        WIDTH / 2,
        3 * HEIGHT / 10,
        200,
        50,
        lambda : False,
        my_font,
        WHITE,
    )

    exit_to_level_select_button = Button(
        "options",
        WIDTH / 2,
        5 * HEIGHT / 10,
        200,
        50,
        lambda : False,
        my_font,
        WHITE,
```

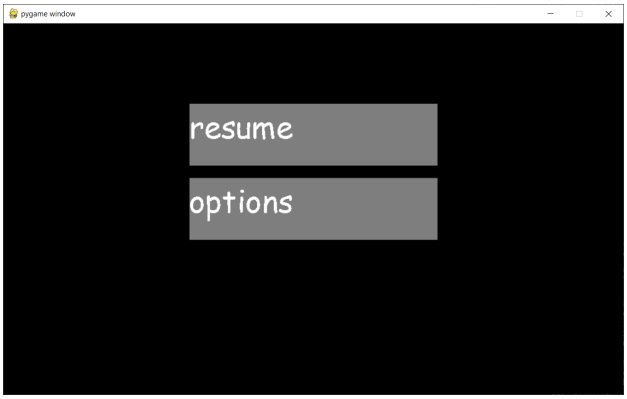
```
    )

    running = True
    while running:

        dt = clock.tick(FPS) * 0.001

        mouse_pos = Vector.from_tuple(pygame.mouse.get_pos())

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return True

            # check all buttons for mouse down
            if event.type == pygame.MOUSEBUTTONDOWN:
                resume_button.check_mouse_down(mouse_pos)
                exit_to_level_select_button.check_mouse_down(mouse_pos)

            # check all buttons for mouse up
            if event.type == pygame.MOUSEBUTTONUP:
                if resume_button.check_mouse_up(mouse_pos):
                    return True
                if
exit_to_level_select_button.check_mouse_up(mouse_pos):
                    return True

            if event.type == pygame.KEYDOWN:
                # go back if Esc clicked
                if event.key == pygame.K_ESCAPE:
                    return

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        resume_button.draw(screen)
        exit_to_level_select_button.draw(screen)

        pygame.display.flip()
```

Test:

```
WIDTH = 1000
HEIGHT = 600
```

```
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.font.init()

pause_menu(screen)

# this will only run when all the screens have finished
pygame.quit()
```

Output:



However, the buttons don't close the screen, or do anything at all. This is because so far, all a button can do is open a screen and run it until it finishes. The only externality is whether the whole game will then close afterwards. To give the buttons more functionality, the button's on click function must return a number controlling how many screens should be closed after the process is finished. -1 will be returned if all the screens should be closed.

Fig 27.1

So I will change all the other screens to use this system.

To exit all screens when the cross is clicked.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        return -1
```

To exit the current screen when the escape button is pressed.

```
if event.type == pygame.KEYDOWN:
    # go back if Esc clicked
    if event.key == pygame.K_ESCAPE:
        return 0
```

To decrement the return value when receiving a number from another screen.

```
screens_to_exit = level_button.check_mouse_up(mouse_pos)
if screens_to_exit:
    return screens_to_exit - 1
```

Output:
The old screens are exited correctly.

To make the pause menu work, the resume button should return 1 to go back to the level. And the exit to level select button should return 2 to exit another screen and therefore go back to the level select screen.

```
resume_button = Button(
    "resume",
    WIDTH / 2,
    3 * HEIGHT / 10,
    200,
    50,
    lambda : 0,
    my_font,
    WHITE,
)

exit_to_level_select_button = Button(
    "options",
    WIDTH / 2,
    5 * HEIGHT / 10,
    200,
    50,
    lambda : 1,
    my_font,
    WHITE,
)
```

Test:

```
WIDTH = 1000
HEIGHT = 600

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.font.init()

pause_menu(screen)

# this will only run when all the screens have finished
pygame.quit()
```

Output:

Then the screen closes when I press one of the buttons.

Then to implement this screen into the gameplay, I add this to the gameplay function

```
if event.key == pygame.K_ESCAPE:
    screens_to_exit = pause_menu(screen)
    if screens_to_exit:
        return screens_to_exit - 1
```

I also change the text on the exit to level select button.

```
exit_to_level_select_button = Button(
    "exit_to_level_select",
    WIDTH / 2,
    5 * HEIGHT / 10,
    200,
    50,
    lambda : 2,
    my_font,
    WHITE,
)
```

Test:

```
WIDTH = 1000
HEIGHT = 600

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.font.init()

level_select(screen, User.example())

# this will only run when all the screens have finished
pygame.quit()
```

Output:

```
  File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\gameplay.py", line 13, in <module>
    from .pausemenu import pause_menu
  File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\pausemenu.py", line 9, in <module>
    from .levelselect import level_select
  File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\levelselect.py", line 11, in <module>
    from .gameplay import gameplay
ImportError: cannot import name 'gameplay' from partially initialized module 'pygameframes.gameplay' (most likely due to a circular import) (D:\usb for win down backup (goat)\GOATwvenv\pygameframes\gameplay.py)
```

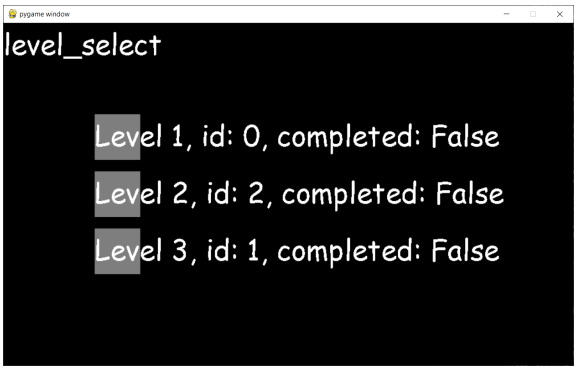
(Logic Error)

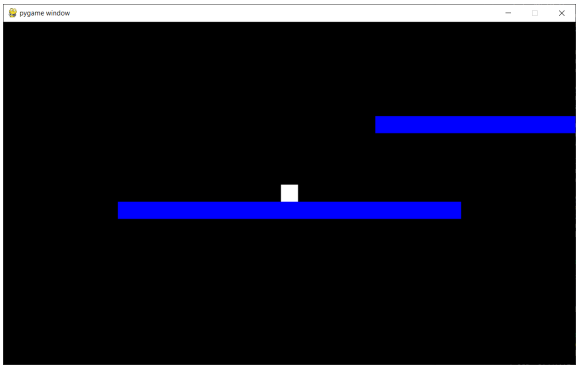
This happens because I accidentally left in an unused import which caused infinite recursive imports.

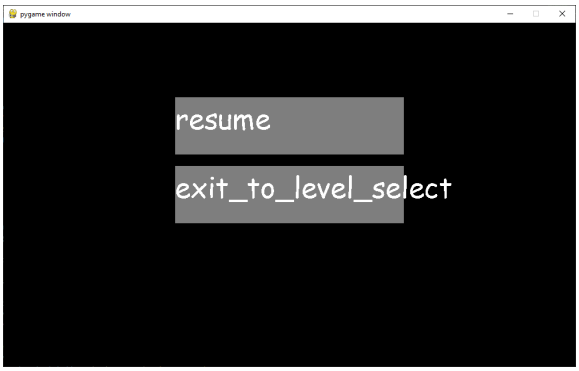
Fig 27.2

I removed them.

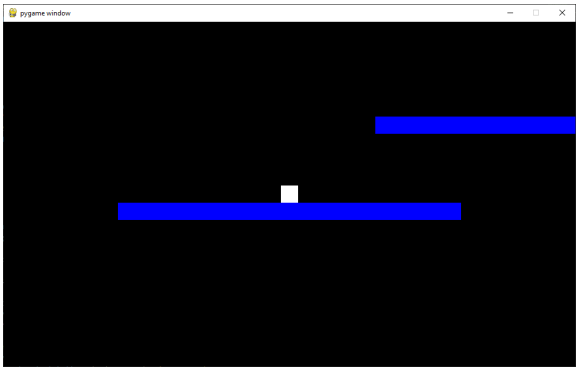
Output:

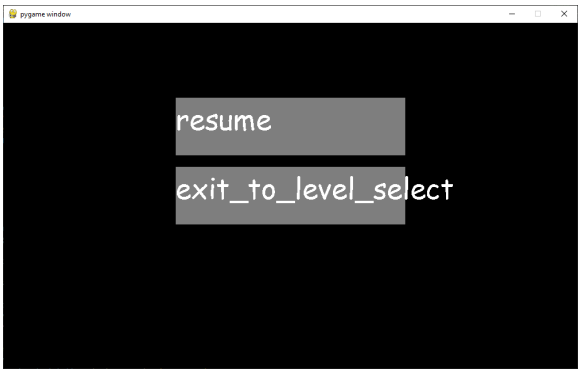




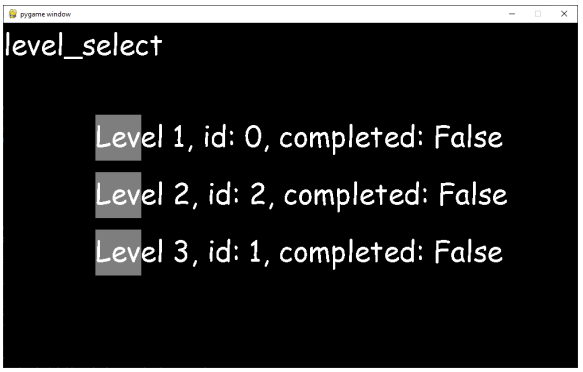


Then after clicking resume





Then after clicking exit_to_level_select



Test number	Test Description	Test data (if applicable)	Justification	Outcome
27.1	Do all the necessary buttons show up when running the pause screen?	n/a	All the buttons on the pause screen must be shown so that the user can navigate and control the program.	As expected
27.2	Do the buttons on the pause screen exit the pause screen?	n/a	When running the pause screen on its own without any parent screen, the screen should close because both buttons will take the user out of the screen and back to a parent screen.	Failed and fixed by making screen functions return a number corresponding to the number of extra screens to return out of e.g. a return value of 2 go back two screens. This makes the functionality of the screen stack much more powerful (**Fig 27.1**).
27.3	Is the pause screen correctly implemented into the other gameplay functions?	n/a	This will test further that the number of screens to exit works properly and the buttons on the pause screen must allow the user to navigate correctly.	Failed and fixed by removing a recursive import (**Fig 27.2**)

Progression

To store progression, on each user object, I will store a dictionary that is structured as the following

```
{
        <level_id>: {
                “completed”: <boolean>,
                “player_x”: <number>,
                “player_y”: <number>,
                “player_vel_x”: <number>,
                “player_vel_y”: <number>,
        },
        etc,
}
```

The user class needs an attribute for this progression.
I will also add a function to return an example user for testing.

```
class User:
    def __init__(self, username, password):

        self.username = username
        self.password = password
        self.progression = {}

    def __str__(self):
        return f'username: {self.username}, password: {self.password},
progression: {self.progression}'

    @staticmethod
    def example():
        return User("firstname", "surname")
```

To save progression, I must give the gameplay functions access to the current user object. To do this, all the gameplay functions (these are main menu, level select and gameplay) must take a user object as an argument and mutate it as the user plays and completes levels. This object will then be saved to secondary storage when a save or save and exit button is clicked.
So I changed the functions and pass the user object to them whenever they are called.
I also changed the demos/tests for each of the screens to use the example user.
Then I pass the user into the main gameplay function that is called when the user logs in

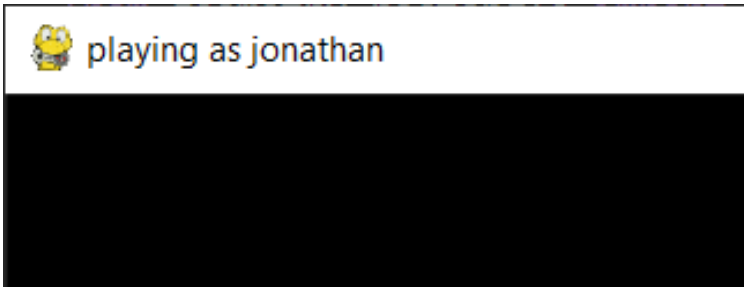
```
def on_login(user):
    WIDTH = 1000
    HEIGHT = 600
```

```
    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))

    pygame.font.init()

    main_menu(screen, user)

    # this will only run when all the screens have finished
    pygame.quit()
```

To test that the user is being passed in properly, I use this line of code

```
    pygame.display.set_caption(f"playing as {user.username}")
```

Output:



To save the user's progression, I have to change the progression attribute on the user object when the user completes a level.

```
if player.rectangle_collision_detection(level.target):
    print("level completed")
    user.progression[level.level_id].completed = True
```

To test that this is being assigned properly, I will display whether a level has been completed or not on the level select screen like this

```
# create the level buttons
level_buttons = [
    Button(
        f"Level {i + 1}, id: {levels[i].level_id}, completed:
{user.progression[level.level_id].completed}",
        200,
        200 + 100 * i,
        40,
        40,
        create_level_function(level),
        # (lambda level: lambda: gameplay(screen, level))(level),
        my_font,
        WHITE,
    )
    for i, level in enumerate(levels)
```

```
]
```

Output:

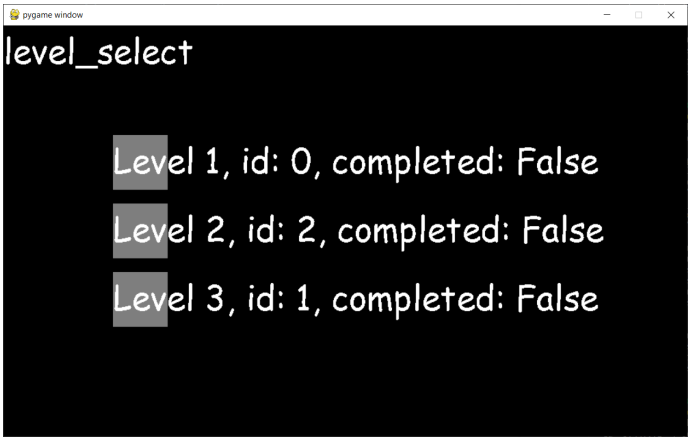
File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\levelselect.py", line 45, in <listcomp>
f"Level {i + 1}, id: {levels[i].level_id}, completed: {user.progression[level.level_id].completed}",
KeyError: '0'
(Logic Error)

Fig 28.1

This happens because the progression attribute does not have the level id as a key.
To fix this, I will make sure that if it doesn't have a key, it will display false.

```
# create the level buttons
level_buttons = [
    Button(
        f"Level {i + 1}, id: {level.level_id}, completed:
{user.progression[level.level_id].completed if
hasattr(user.progression, level.level_id) else False}",
        200,
        200 + 100 * i,
        40,
        40,
        create_level_function(level),
        # (lambda level: lambda: gameplay(screen, level))(level),
        my_font,
        WHITE,
    )
    for i, level in enumerate(levels)
]
```

Output:



Test:

Loading up the level select screen as the example user, selecting a level.

Output:

```
  File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\gameplay.py", line 73,
in gameplay
    user.progression[level.level_id].completed = True
KeyError: '1'
```

(Logic Error)

This happens because the user.progression[“1”] does not exist as it has not been initialised yet.

Fig 28.2

To fix this, I will initialise the progression structure when the user enters the level.

```
def gameplay(screen, level, user):

    user.progression[level.level_id] = {}

    ...
```

Test:

Loading up the level select screen as the example user, selecting a level and completing it.

Output:

```
  File "D:\usb for win down backup (goat)\GOATwvenv\pygameframes\gameplay.py", line 75,
in gameplay
    user.progression[level.level_id].completed =
True
AttributeError: 'dict' object has no attribute 'completed'
```

(Logic Error)

This happens because I used the object attribute notation, not the dictionary notation.

Fig 28.3

I should probably have a progression object anyway, so I’ll create that.

```
class LevelProgression:
    def __init__(self, level):
        self.completed = False
        self.player_pos = level.start_pos
        self.player_vel = Vector(0, 0)
```

Then when a level is loaded, a new level progression object will be created for that level, if one doesn’t already exist. The hasattr method doesn’t work properly either, so I’ll us the in keyword.

```
def gameplay(screen, level, user):

```

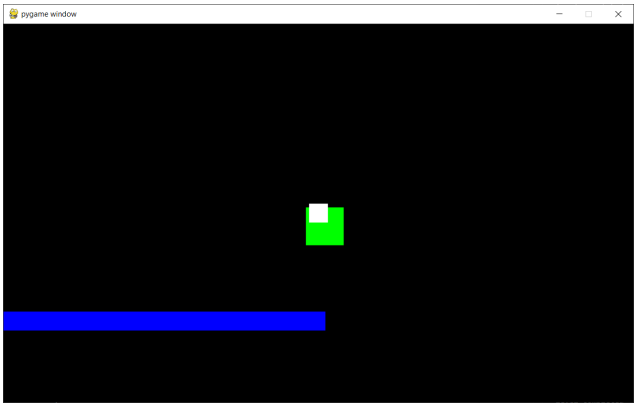
```
    if level.level_id not in user.progression:
        user.progression[level.level_id] = LevelProgression(level)

    ...
```

And I'll also change the level select screen.

```
# create the level buttons
level_buttons = [
    Button(
        f"Level {i + 1}, id: {level.level_id}, completed:
{user.progression[level.level_id].completed if level.level_id in
user.progression else False}",
        200,
        200 + 100 * i,
        40,
        40,
        create_level_function(level),
        # (lambda level: lambda: gameplay(screen, level))(level),
        my_font,
        WHITE,
    )
    for i, level in enumerate(levels)
]
```

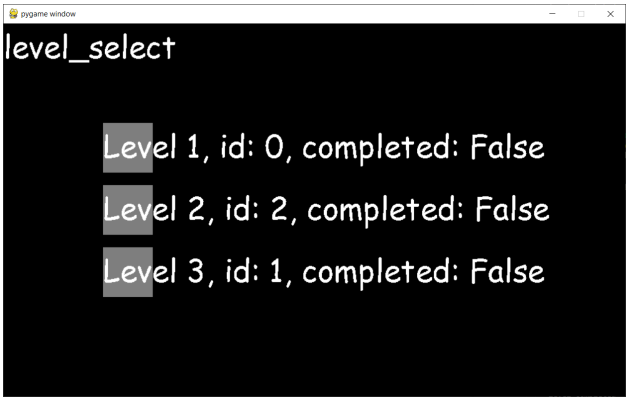
Test:

Loading up the level select screen as the example user, selecting a level and completing it.



Output:

After going back to the level select screen.



(Logic Error)

This happened because when I return to the level select screen, the text on the buttons is not updated. However, the player never actually clicked save (as there is no option for that), so it's sort of intended behaviour.

To implement a way for the player to save, I will add a button to the pause menu.

```
save_button = Button(
    "save",
    WIDTH / 2,
    7 * HEIGHT / 10,
    200,
    50,
    lambda : 1,
    my_font,
    WHITE,
)
```

And check for clicks with the other buttons

```
# check all buttons for mouse down
if event.type == pygame.MOUSEBUTTONDOWN:
    ...
    save_button.check_mouse_down(mouse_pos)
```

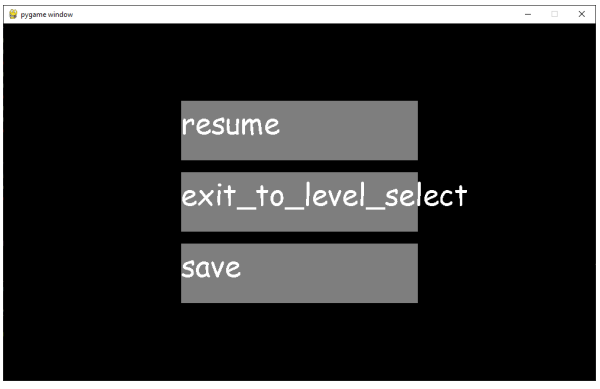
And

```
screens_to_exit = save_button.check_mouse_up(mouse_pos)
if screens_to_exit:
    return screens_to_exit - 1
```

And display it

```
save_button.draw(screen)
```

Output:



Currently, the save button doesn't do anything except exit because I haven't changed its on click function

To save the progress of the player, I will continually modify the player's progression dictionary, but only save this to secondary storage when the player clicks the save button.

so I will create a function that saves the users list to storage, this will save the user that is logged in because the user that logged in is passed by reference to the gameplay and menu functions. Then I will pass this save_users function to all the screens.

```
def save_users():
    save_obj(users, USERS_FILE_PATH)
```

And

```
def on_login(user):
    WIDTH = 1000
    HEIGHT = 600

    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption(f"playing as {user.username}")

    pygame.font.init()

    main_menu(screen, user, save_users)

    # this will only run when all the screens have finished
    pygame.quit()
```

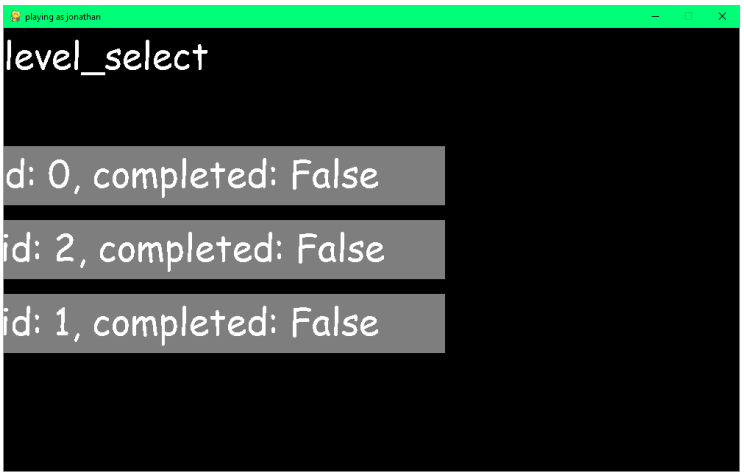
Then to use this function, it will be called when the user clicks the save button in the pause menu.

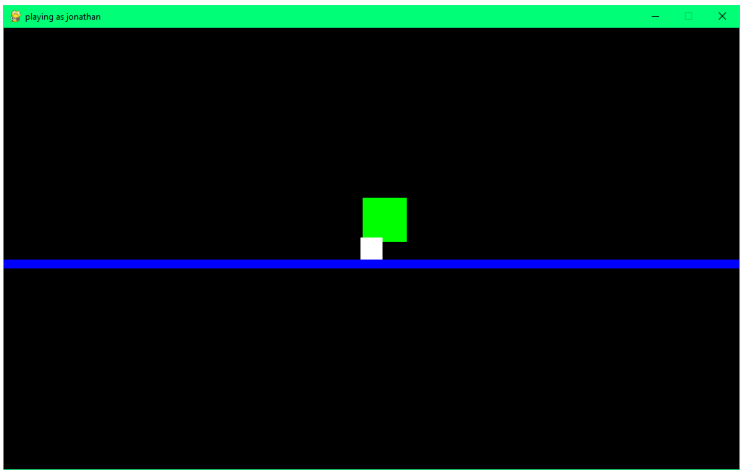
```
def save_button_on_click():
    on_save()
    return 0

save_button = Button(
    "save",
    WIDTH / 2,
    7 * HEIGHT / 10,
    200,
    50,
    save_button_on_click,
    my_font,
    WHITE,
)
```

users file before logging in (only the section for one user):

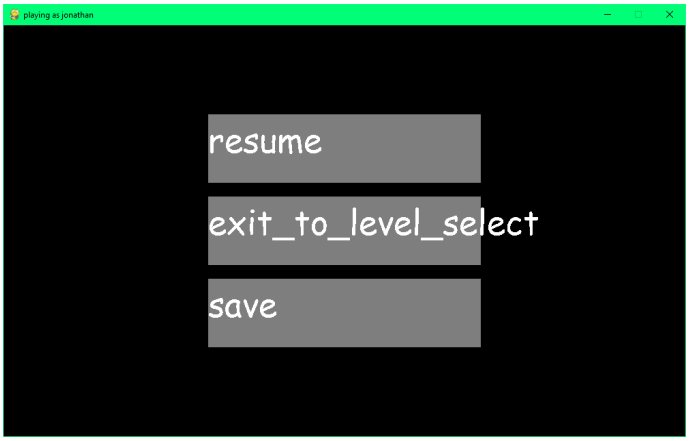
```
}◆ubh)◆◆}◆(h◆jonathan◆h◆password◆h
```

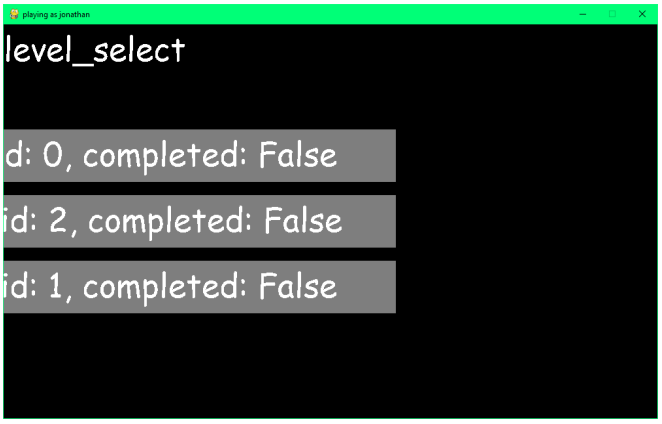
Output:





level completed
level completed
level completed
level completed
level completed
level completed
level completed





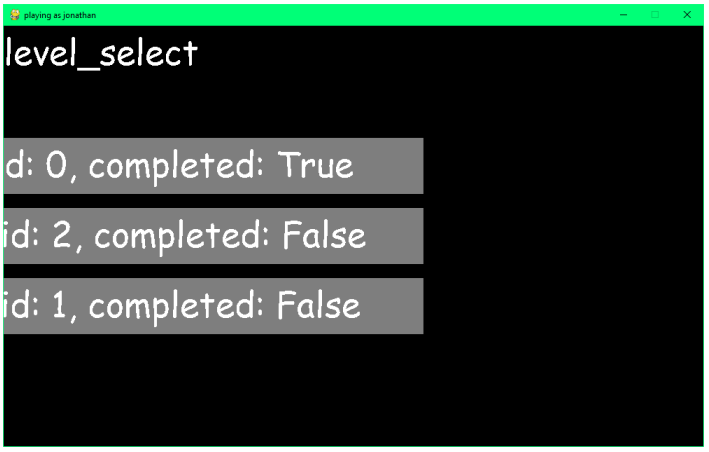
the contents of the users file now (for that one user):

```
}◆ubh)◆◆}◆(h◆jonathan◆h◆password◆h
}◆◆0◆◆levels.levelprogression◆◆LevelProgression◆◆◆)◆◆}◆(◆
completed◆◆◆
player_pos◆◆geometry.vector◆◆Vector◆◆◆)◆◆}◆(◆x◆K◆◆y◆K◆ub◆
player_vel◆h )◆◆}◆(h#K◆h$K◆ububsube.
```

There is now much more info about the user and where they are in the level and that they have completed it.

But it doesn't show up on the level_select screen because the UI hasn't been updated even though the data in the user object has.

On reloading the game with the same user



To have this update when exiting to this screen from the gameplay (without reloading the game), I have to change the text on the buttons when returning to this screen from a level.

Fig 28.4

To achieve this, I have to use this condition

```
if screens_to_exit is not None:
```

This is True if the button has been clicked and the program is now returning to the screen.

This is the procedure I will use to update the text on the level buttons

```
def update_level_buttons_text(levels, level_buttons):
    for i, (level, level_button) in enumerate(zip(levels,
level_buttons)):
        level_button.text = f"Level {i + 1}, id: {level.level_id},
completed: {user.progression[level.level_id].completed if
level.level_id in user.progression else False}"
```

Then I will call it when returning to this screen.

```
# reload the buttons to show the progression when this screen is
returned to from the gameplay
if screens_to_exit is not None:
    update_level_buttons_text(levels, level_buttons)
```

Output:
It doesn't work. (Hard to show it not working as it just looks the same).

This is because the buttons render the text when they are created. I could create a set_text method, but it's probably better to just create the buttons again, as then I can just use this method to create the buttons in the first place.

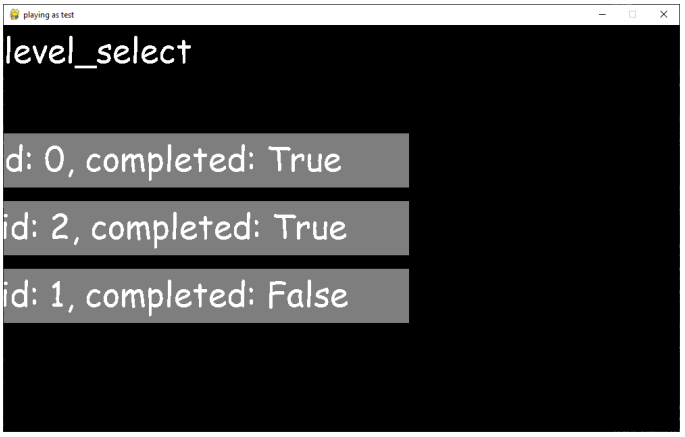
28.5
So I change the update function and the creation to this

```
# function to create the level buttons
def create_level_buttons():
    return [
        Button(
            f"Level {i + 1}, id: {level.level_id}, completed:
{user.progression[level.level_id].completed if level.level_id in
user.progression else False}",
            200,
            200 + 100 * i,
            400,
            40,
            create_level_function(level),
            # (lambda level: lambda: gameplay(screen, level))(level),
            my_font,
            WHITE,
        )
        for i, level in enumerate(levels)
    ]

# create the level buttons
level_buttons = create_level_buttons()
```

and the updating section of code to this

```
# reload the buttons to show the progression when this screen is
returned to from the gameplay
if screens_to_exit is not None:
    level_buttons = create_level_buttons()
```

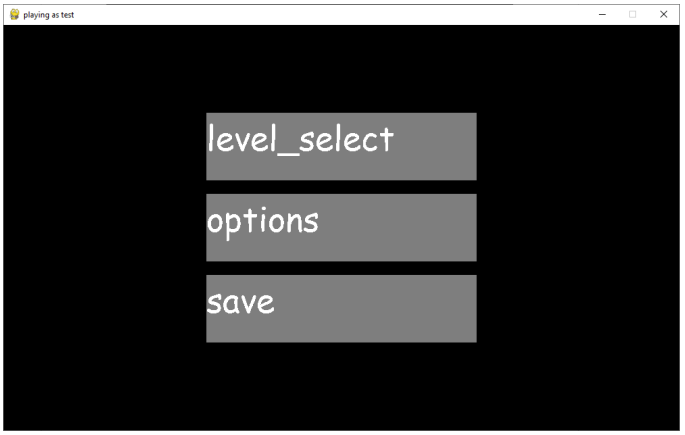
Output:

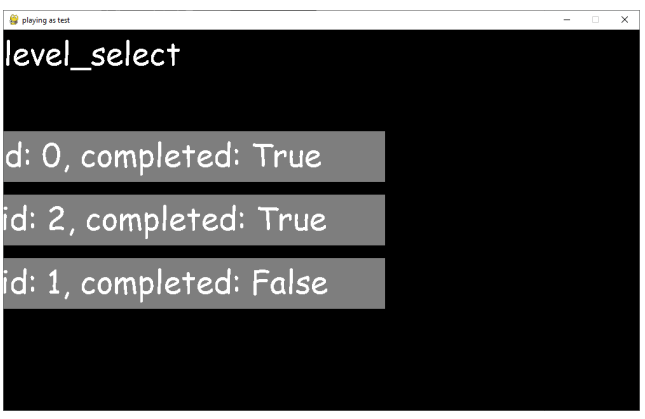


At the moment, the progress shows up on the level select screen even if you don't click save and then doesn't show up if you close and then open the game again without clicking the save button. This is unintuitive as it appears to have been saved, but it hasn't actually been saved to secondary storage. To change this, I will remove the current save button, but instead have a button on the main menu that says save progression and a screen that pops up when you close the game without saving, prompting you to save your progress.

To have a save button on the main menu, I'll just move the save button code there.

Output:





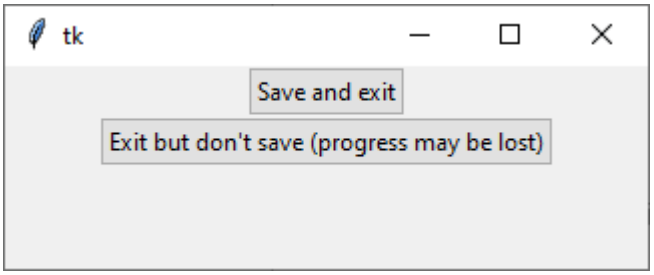
The progress is saved when I click the save button.

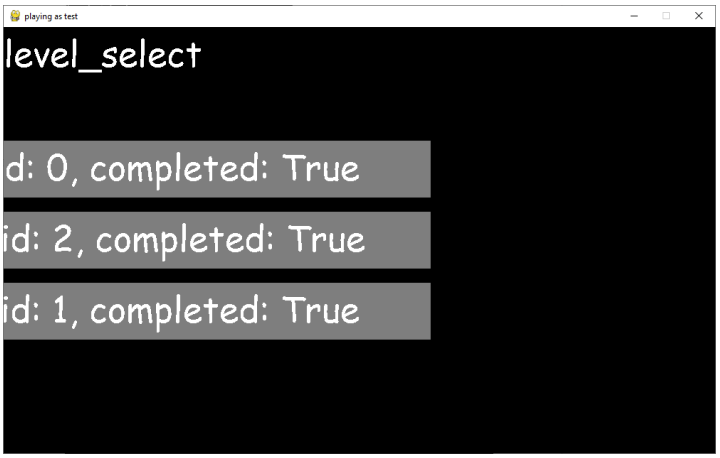
To make the save prompt pop-up. I will use a tkinter GUI screen when the user exits the game. I won't put this part in any of the test demos.

```
def on_login(user):

    ...

    # save progression pop-up
    OptionsScreen([
        Option("Save and exit", save_users),
        Option("Exit but don't save (progress may be lost)", lambda:
None)
    ]).mainloop()

    # this will only run when all the screens have finished
    pygame.quit()
```

Putting the pop-up before the pygame.quit() will mean that the pop-up will open and the game won't close until the pop-up is dealt with which means it will be more obvious for a user that they have to do something.

Output:



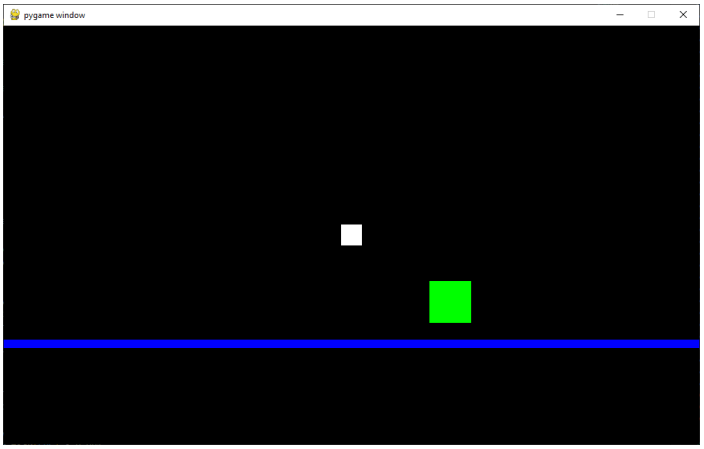


There is also a problem with the pause screen where it counts the time in the pause screen as part of the time of a given frame in pygame. So delta_time is very large when exiting the pause menu. This results in some weird behaviour. To fix this, I will change the delta time

code so that it is set to 0 at the start of the frame, then if Esc is not pressed, it will be ticked as usual, but if esc is pressed, it adds the frame's time to delta time then resets the clock after the pause screen is processed, so the clock is unaware of the time in the pause screen.

```
dt = 0

screen_was_entered = False

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        return -1

    if event.type == pygame.KEYDOWN:
        # go to the pause menu if Esc pressed
        if event.key == pygame.K_ESCAPE:
            dt += clock.tick(FPS) * 0.001
            screens_to_exit = pause_menu(screen, on_save)
            if screens_to_exit:
                return screens_to_exit - 1
            screen_was_entered = True
            clock.tick()

if not screen_was_entered:
    dt += clock.tick(FPS) * 0.001
```

Output:



(There is no more glitchy movement when exiting the pause screen.)

It can be simplifies to this, and it does the same thing

```
dt = clock.tick(FPS) * 0.001

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        return -1

    if event.type == pygame.KEYDOWN:
        # go to the pause menu if Esc pressed
        if event.key == pygame.K_ESCAPE:
            screens_to_exit = pause_menu(screen, on_save)
            if screens_to_exit:
                return screens_to_exit - 1
            clock.tick()
```

Test number	Test Description	Test data (if applicable)	Justification	Outcome
28.1	Is user data passed to the pygame screen correctly?	The user data (specifically the username e.g. “jonathan”) (valid)	The user data must be passed in to correctly load progression.	As expected
28.2	Does the level select screen correctly display whether the user has completed the level.	The boolean representing whether the player have complete the level or not, False (valid)	The player should be able to see which levels they have completed so that they know which ones they would like to play next.	Failed and fixed by adding a default value if the level progression dictionary doesn’t have one of the necessary keys (**Fig 28.1**).
28.3	Is the level progression loaded correctly when entering a level?	The level data (user.progression[“1”]) (valid)	The progression needs to be loaded into the level correctly to know where the player should be and other elements of progression.	Failed and foxed by initialising the level progression to an empty dictionary if it doesn’t already exist (**Fig 28.2**).

28.4	Is the level progression updated when the user hits the target?	n/a	The progression needs to be updated when the user completes the level so that the user can be told which levels they have completed in the future.	Failed and fixed by switching to a level progression class to store the level progression data (**Fig 28.3**).
28.5	Does the updated level progression update the level select screen with the new information?	The boolean representing whether the player have complete the level or not, True (valid)	The user needs to be able to see which levels they have completed.	Failed and fixed by updating the level select screen when returning to it (**FIg 28.4**). And re-rendering the button text (**Fig 28.5**).
28.6	Does a prompt to save to secondary storage pop up and function properly when the user tries to exit the game?	n/a	The user may exit the game without saving by accident and lose their progress.	As Expected
28.7	Does a save button appear and function correctly on the main menu?	n/a	The user may want to save manually without the game closing.	As Expected

Sound Effects

https://www.pygame.org/docs/ref/mixer.html

I want a sound effect to play when the user jumps. I'll get a sound effect and load it into a level, create a function to play is and then pass that function into the player's jump function.

```
jump_sound = pygame.mixer.Sound("jumpsound.wav")
def play_jump_sound():
    jump_sound.play()
```

```
player.jump(jump_input=keys[pygame.K_SPACE], power=35, dt=dt,
play_jump_sound=play_jump_sound)
```

```
def jump(self, jump_input, power, dt, play_jump_sound=None):
    if jump_input and self.grounded:
        self.apply_acc(Vector(0, -power / dt))
        # play the jump sound
        if play_jump_sound:
            play_jump_sound()
```

Output:
(It works)

Test number	Test Description	Test data (if applicable)	Justification	Outcome
29.1	Does the game play a sound effect when the player jumps?	The name of the sound file to be used, "jumpsound. wav" (valid).	The game should play a sound when the player jumps to make the game more responsive.	As expected

Game Over Screen and fail conditions

To create a game over screen, I need a game over condition. For now, this will be a bounds rectangle, such that if the player leaves the rectangle, game over will be triggered. I can add a kill-box later. This does not need to have continuous collision because it won't matter if the player goes out of bounds for a frame or two before being killed.

To achieve this, I will add a bounds rectangle to the levels.
I have already done this in the JSON file here

```
"bounds": {
    "x": 0,
    "y": 0,
    "hw": 100,
    "hh": 100
},
```

In the python class here

```
self.bounds = bounds
```

And in the level loading here

```
bounds = Rectangle(
    level_dict["bounds"]["x"],
    level_dict["bounds"]["y"],
    level_dict["bounds"]["hw"],
    level_dict["bounds"]["hh"],
)
```

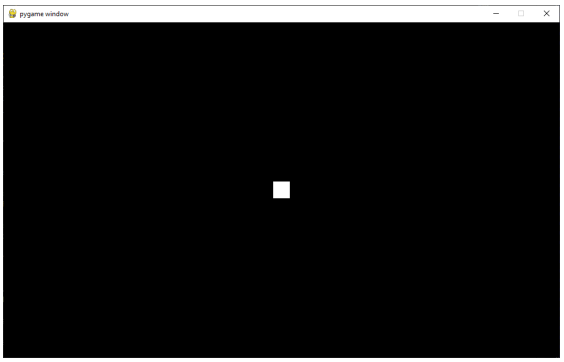
Now I'll add a method to the player class to check whether the player is in bounds

```
def is_in_bounds(self, bounds: Rectangle):
    if self.rect.pos.x > bounds.right_pos():
        return False
    if self.rect.pos.y > bounds.bottom_pos():
        return False
    if self.rect.pos.x < bounds.left_pos():
        return False
    if self.rect.pos.y < bounds.top_pos():
        return False
    return True
```

Then I'll call it in the gameplay loop

```
if not player.is_in_bounds(level.bounds):
    print("out of bounds")
```

Output:



out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds
out of bounds

Now I'll create a game over screen

```
def game_over(screen):
    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()

    my_font = pygame.font.SysFont("Comic Sans MS", 50)
    text_render = my_font.render("game_over", False, WHITE)

    retry_button = Button(
        "retry",
        WIDTH / 2,
```

```
        3 * HEIGHT / 10,
        200,
        50,
        lambda: 1,
        my_font,
        WHITE,
    )

    exit_to_level_select_button = Button(
        "exit_to_level_select",
        WIDTH / 2,
        5 * HEIGHT / 10,
        200,
        50,
        lambda: 2,
        my_font,
        WHITE,
    )

    running = True
    while running:
        dt = clock.tick(FPS) * 0.001

        mouse_pos = Vector.from_tuple(pygame.mouse.get_pos())

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return -1

            # check all buttons for mouse down
            if event.type == pygame.MOUSEBUTTONDOWN:
                retry_button.check_mouse_down(mouse_pos)
                exit_to_level_select_button.check_mouse_down(mouse_pos)
                #// save_button.check_mouse_down(mouse_pos)

            # check all buttons for mouse up
            if event.type == pygame.MOUSEBUTTONUP:
                screens_to_exit =
retry_button.check_mouse_up(mouse_pos)
                if screens_to_exit:
                    return screens_to_exit - 1
                screens_to_exit =
exit_to_level_select_button.check_mouse_up(mouse_pos)
```

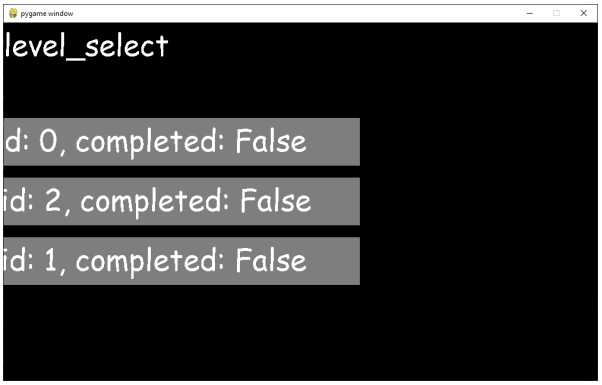
```
                if screens_to_exit:
                    return screens_to_exit - 1
                #// screens_to_exit =
save_button.check_mouse_up(mouse_pos)
                #// if screens_to_exit:
                #//     return screens_to_exit - 1

            if event.type == pygame.KEYDOWN:
                # go back if Esc clicked
                if event.key == pygame.K_ESCAPE:
                    return 0

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)

        # draw the game over text
        screen.blit(text_render, (0, 0))

        # draw the buttons
        retry_button.draw(screen)
        exit_to_level_select_button.draw(screen)
        #// save_button.draw(screen)

        pygame.display.flip()
```

And replace the out of bounds code with

```
if not player.is_in_bounds(level.bounds):
    screens_to_exit = game_over(screen)
    if screens_to_exit:
        return screens_to_exit - 1
```

Output:
When going out of bounds

When clicking exitt_to_level_select



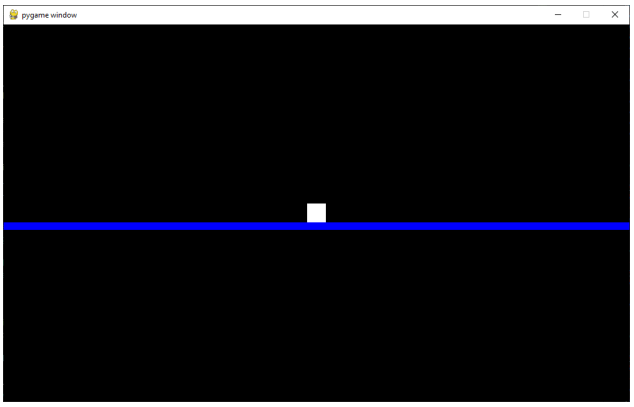
When clicking retry (the game over screen pops straight back up again)

(Logic Error)
This happens because as soon as the program returns to the level, the player is already out of bounds, so the game over state is triggered again.

Fig 30.1
To fix this, I will reset the level after returning to the gameplay from the game over screen.

```
if not player.is_in_bounds(level.bounds):
    screens_to_exit = game_over(screen)
    if screens_to_exit:
        return screens_to_exit - 1

    # reset the level
    clock = pygame.time.Clock()

    player = Player(level.start_pos.x, level.start_pos.y, 0.5, 0.5,
vel_x=0, vel_y=0)
    camera = Camera(player.rect.pos.x, player.rect.pos.y, WIDTH,
HEIGHT, game_space_visible_height=20)

    continue
```

Output:
After clicking retry



The player goes back to the starting position.

Now to add a kill box I'll add a list of them to the levels
In the JSON file

```
"kill_boxes": [
    {
        "x": 2,
        "y": -1,
        "hw": 1,
        "hh": 0.2
    }
]
```

In the level python class

```
class Level:
    def __init__(self, level_id, start_pos, target, bounds, platforms,
kill_boxes):
        self.level_id = level_id
        self.start_pos = start_pos
        self.target = target
        self.bounds = bounds
        self.platforms = platforms
        self.kill_boxes = kill_boxes
```

When loading levels

```
# Create the kill boxes (a list of rectangles)
kill_boxes = []
for platform_dict in level_dict["kill_boxes"]:
    kill_box = Rectangle(
        platform_dict["x"],
        platform_dict["y"],
        platform_dict["hw"],
        platform_dict["hh"],
    )
    kill_boxes.append(kill_box)
```

The player interacting with them

```
if player.rectangle_collision_detection(level.target):
    screens_to_exit = game_over(screen)
    if screens_to_exit:
        return screens_to_exit - 1

    # reset the level
    clock = pygame.time.Clock()

    player = Player(level.start_pos.x, level.start_pos.y, 0.5, 0.5,
vel_x=0, vel_y=0)
    camera = Camera(player.rect.pos.x, player.rect.pos.y, WIDTH,
HEIGHT, game_space_visible_height=20)

    continue
```

Drawing them

```
# draw the kill boxes
for k in level.kill_boxes:
    pygame.draw.rect(
        screen,
        RED,
        camera.game_space_to_screen_space_corner_rect_tuple(
            k.corner_rect_tuple()
        ),
    )
```

Output:

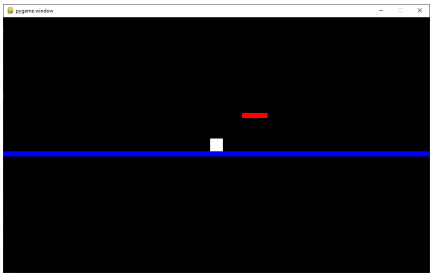
```
  File "C:\Users\jonathan\school\cs\GOAT\levels\loadlevels.py", line 42, in load_levels
    for platform_dict in level_dict["kill_boxes"]:
                         ~~~~~~~~~~^^^^^^^^^^^^^^^
KeyError: 'kill_boxes'
```

(Logic Error)

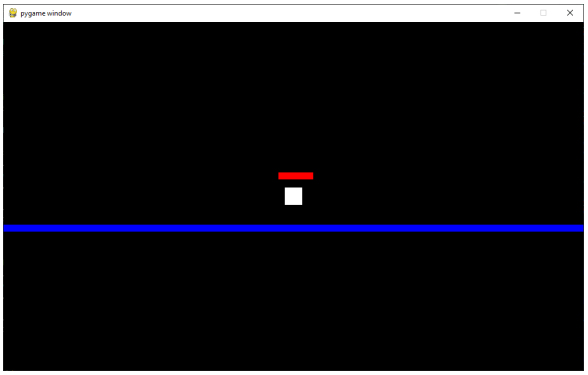
Fig 30.2

This happened because I only added a kill boxes list to the first level.
After adding an empty kill boxes list to the JSON file.

Output:



Then after hitting the kill box



Nothing happens (Logic Error)

Fig 30.3

This happens because I forgot to update the player interaction with the kill boxes code to use the kill boxes instead of the target

```
# check whether the player has hit a kill box
hit_kill_box = False
for k in level.kill_boxes:
    if player.rectangle_collision_detection(k):
        hit_kill_box = True

# go to game over screen if it has hit a kill box
if hit_kill_box:
    screens_to_exit = game_over(screen)
    if screens_to_exit:
        return screens_to_exit - 1

    # reset the level
    clock = pygame.time.Clock()

    player = Player(level.start_pos.x, level.start_pos.y, 0.5, 0.5,
vel_x=0, vel_y=0)
    camera = Camera(player.rect.pos.x, player.rect.pos.y, WIDTH,
HEIGHT, game_space_visible_height=20)

    continue
```

Output:

After hitting the kill box

Test number	Test Description	Test data (if applicable)	Justification	Outcome
30.1	Can the game detect when the player is out of bounds?	The level bounds e.g. (x:0, y:0, hw: 100, hh: 100) (valid)	The game needs to detect when the player goes out of bounds so it can be killed.	As expected
30.2	Does the game over screen get rendered properly and get triggered properly when the player goes out of bounds?	n/a	The player should be sent to the game over screen when going out of bounds to stop the player falling forever.	As expected
30.3	Does the game restart properly when clicking retry on the game over screen?	n/a	The player should be able to have another go at the level after failing to keep them engaged.	Failed and fixed by resetting the level when the player dies (**Fig 30.1**).

30.4	Can the kill-boxes be loaded into the level correctly?	The positions of the kill-boxes e.g. [
		{
		"x": 2,
		"y": -1,
		"hw": 1,
		"hh": 0.2
		}
		] (valid)
			This will provide a fail condition so that the player can lose the level, this will give the levels an element of challenge, so they are not too easy and will keep the user engaged.	Failed and fixed by adding a kill box list to all the levels in the levels JSON file (even if it's just an empty list) (**FIg 30.2**).

30.5	Does hitting a kill box kill the player?	The positions of the kill-boxes e.g. [
		{
		"x": 2,
		"y": -1,
		"hw": 1,
		"hh": 0.2
		}
		] (valid)
			This will provide a fail condition so that the player can lose the level, this will give the levels an element of challenge, so they are not too easy and will keep the user engaged.	Failed and fixed by updating the collision code with the kill box(es) (**Fig 30.3**).

Level Complete Screen

When the player completes a level, I want a screen to pop up with a special offer and other options.

So I'll create a win-state screen.

```
def level_complete(screen):
    WIDTH = screen.get_width()
    HEIGHT = screen.get_height()

    FPS = 60

    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)

    clock = pygame.time.Clock()
    # load and create all the media for the level complete screen
    my_font = pygame.font.SysFont("Comic Sans MS", 50)
    text_render = my_font.render("level_complete", False, WHITE)
    special_offer_image = pygame.image.load("assets/specialoffer.png")
    pygame.mixer.Sound("assets/levelcompletesound.wav").play()

    retry_button = Button(
        "retry",
        WIDTH / 2,
        3 * HEIGHT / 10,
        200,
        50,
        lambda: 1,
        my_font,
        WHITE,
    )

    exit_to_level_select_button = Button(
        "exit_to_level_select",
        WIDTH / 2,
        5 * HEIGHT / 10,
        200,
        50,
        lambda: 2,
```

```
        my_font,
        WHITE,
    )

    running = True
    while running:
        dt = clock.tick(FPS) * 0.001

        mouse_pos = Vector.from_tuple(pygame.mouse.get_pos())

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return -1

            # check all buttons for mouse down
            if event.type == pygame.MOUSEBUTTONDOWN:
                retry_button.check_mouse_down(mouse_pos)
                exit_to_level_select_button.check_mouse_down(mouse_pos)
                #// save_button.check_mouse_down(mouse_pos)

            # check all buttons for mouse up
            if event.type == pygame.MOUSEBUTTONUP:
                screens_to_exit =
retry_button.check_mouse_up(mouse_pos)
                if screens_to_exit:
                    return screens_to_exit - 1
                screens_to_exit =
exit_to_level_select_button.check_mouse_up(mouse_pos)
                if screens_to_exit:
                    return screens_to_exit - 1
                #// screens_to_exit =
save_button.check_mouse_up(mouse_pos)
                #// if screens_to_exit:
                #//     return screens_to_exit - 1

            if event.type == pygame.KEYDOWN:
                # go back if Esc clicked
                if event.key == pygame.K_ESCAPE:
                    return 0

        keys = pygame.key.get_pressed()

        screen.fill(BLACK)
```

```
        # draw the level complete text
        screen.blit(text_render, (0, 0))

        # draw the special offer image
        screen.blit(special_offer_image, (50, 50))

        # draw the buttons
        retry_button.draw(screen)
        exit_to_level_select_button.draw(screen)
        #// save_button.draw(screen)

        pygame.display.flip()
```

(I also added a sound.)
And call this function when the player completes the level.

```
if player.rectangle_collision_detection(level.target):
    # // print("level completed")
    user.progression[level.level_id].completed = True

    screens_to_exit = level_complete(screen)
    if screens_to_exit:
        return screens_to_exit - 1

    # reset the level
    clock, player, camera = init_clock_player_camera()

    continue
```

Output:
When hitting the target rectangle

I'll tidy it up later.

Test number	Test Description	Test data (if applicable)	Justification	Outcome
31.1	Is a sound played and a special offer image shown when the user completes a level?	The name of the image and sound files “assets/specialoffer.png” and "assets/levelcompletesound.wav" (valid)	The player should be rewarded when completing a level.	As expected

More Progression

I want to save the user's position in the level, so that they can return and finish it off later. To do this I create a game state object like this which is saved in the level progression

```
class GameState:
    def __init__(self, level):
        self.player = Player(
            level.start_pos.x, level.start_pos.y, 0.5, 0.5, vel_x=0,
vel_y=0
        )
        self.clock = pygame.time.Clock()

    def respawn(self, level):
        self.player = Player(
            level.start_pos.x, level.start_pos.y, 0.5, 0.5, vel_x=0,
vel_y=0
        )
        self.clock = pygame.time.Clock()

    def update(self, clock, player):
        self.clock = clock
        self.player = player

    def get(self):
        return (self.clock, self.player)
```

```
class LevelProgression:
    def __init__(self, level):
        self.completed = False
        self.game_state = GameState(level)
```

I will load the game sate into the level at the start like so

```
clock, player = user.progression[level.level_id].game_state.get()
```

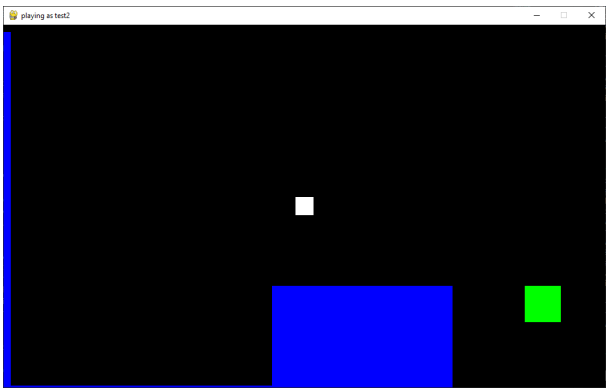
And I will create a method to respawn the player that reloads the game state and uses that to reset the player.

```
def respawn_clock_player():
    user.progression[level.level_id].game_state.respawn(level)
    return user.progression[level.level_id].game_state.get()
```

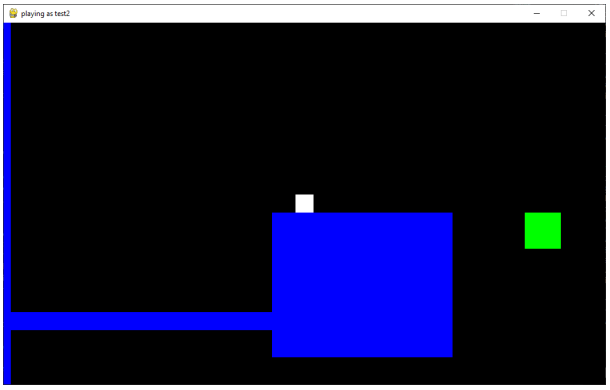
And then I will call this when the player respawns like this

```
clock, player = respawn_clock_player()
```

Test:
Pausing here and clicking exit_to_level_select then resuming the level



Output:



The player is instantly on the ground (Logic Error)

This happens because the clock keeps going while the user is in the level select screen, so delta_time on the first frame is too large.

Fig 32.1
To stop this from happening, I will tick the clock when returning

```
clock, player = user.progression[level.level_id].game_state.get()
clock.tick()
```

Test:
Saving and exiting the game

Output:

```
  File "C:\Users\jonathan\school\cs\GOAT\picklefuncs.py", line 10, in save_obj
    pickle.dump(obj, f)
TypeError: cannot pickle 'pygame.time.Clock' object
```

(Logic Error)

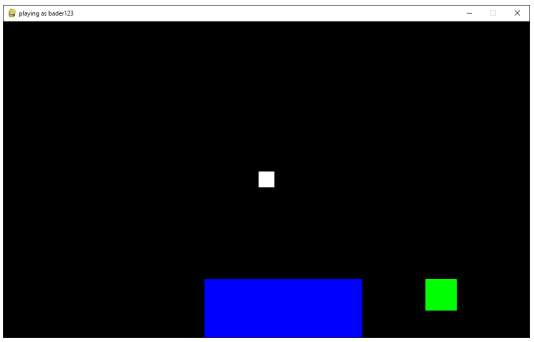
Fig 32.2
I can't save the pygame clock object, and I don't really need to anyway, so I'll just get rid of the clock saving stuff.

```
class GameState:
    def __init__(self, level):
        self.player = Player(
            level.start_pos.x, level.start_pos.y, 0.5, 0.5, vel_x=0,
vel_y=0
        )

    def respawn(self, level):
        self.player = Player(
            level.start_pos.x, level.start_pos.y, 0.5, 0.5, vel_x=0,
vel_y=0
        )

    def update(self, player):
        self.player = player

    def get(self):
        return (self.player)
```

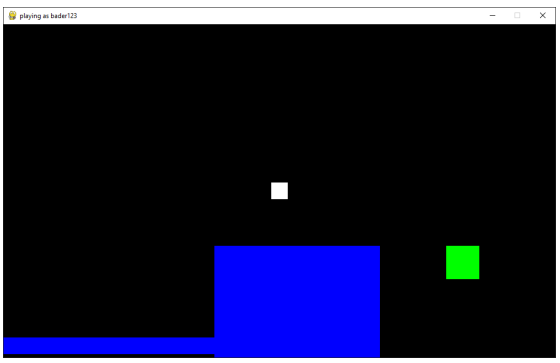
```
player = user.progression[level.level_id].game_state.get()
```

Output:

Just before pausing:



Then when saving exiting and resuming the same level:



(The position and velocity are saved).

Test number	Test Description	Test data (if applicable)	Justification	Outcome
32.1	Is the player's position a velocity saved when exiting a level?	The player's position and velocity e.g. pos = 10, 10 velocity = 5, 20 (valid)	The player should be able to resume where they left off.	Failed and fixed by updating the clock when re-entering a level (**Fig 32.1**) by not saving the clock at all as it is not needed, I can just create a new one when the user enters a level (**Fig 32.2**).

Evaluation

Post-Development Testing

White Box Testing

Function:

In this table, I will describe, justify and show evidence for the outcome of tests to analyse the functionality of my solution. I will do this to measure to what extent the functionality of my solution satisfies the requirements of the solution and the plan of how these requirements should be implemented.

Functionality Test #	Description	Test data (if applicable)	Justification	Expected outcome	Actual Outcome	Evidence
1	Does the password checker reject and accept and reject the right passwords?	“abc” (erroneous), “hello” (boundary), “password123” (valid)	Passwords must be rejected if their length is less than 5 (or some other number) to ensure security.	Passwords shorter than 5 characters should be rejected.	As expected	**Fig E.1**, **Fig E.2**
2	Can the text entries be edited, and can the changes be checked?	Typing in the text entry while it is selected. (valid) Typing while the text entry is not selected (boundary)	For the user to enter their details, I must be able to check the text entries’ contents when the submit button is pressed.	The user should be able to type in the entries and then I should see visible evidence that the details entered are being checked.	As Expected	**Fig E.3**, **Fig E.4**
3	Does clicking the submit button on the create account screen create and save a user to a file in secondary storage.	Clicking the button with valid details. (valid) Clicking the button with invalid details. (boundary)	A new user should be saved to a file in secondary storage when the button is clicked so that the user can login later, but only when the details in the text entries are valid.	The data in the users binary file should be changed to add a new user when the user clicks the create account button.	As expected	**Fig E.5**, **Fig E.6, Fig E.7**

4	Does the program load the users from the file in secondary storage into main memory?	n/a	The program needs access to the users' data to check entered login details against.	The users should be loaded from the file into a list of user objects. A visible effect of this is that the logged in username will be visible on the game screen.	As expected	**Fig E.8**
5	Is the correct user selected when the user enters their login details.	Username and password correct. (valid) Username correct but password incorrect. (boundary) Username not in the list of users and any password. (erroneous)	The entered username in the login screen must be used to select a user from the list of users with the same username (if there is one). And then check that the passwords match. This will ensure that users can only get access to their own account (the one they know the username and password for).	The username will match the username entered when logging in.	As expected	**Fig E.9, Fig E.10**
6	Do the buttons correctly call their on click function when clicked, and do nothing when not clicked.	Clicking on the button. (valid) Holding down the button, moving out of the button and then releasing. (boundary) Clicking outside the button, moving inside the button and then releasing. (boundary)	The buttons must only activate when the user both clicks inside the button and releases inside the button. This is because this is the most common way buttons work in other programs, so it will be intuitive.	When clicking the level select button on the main menu, the user should be taken to the level select screen.	As expected	**Fig E.11, Fig E.12**
7	Are the platforms displayed	n/a	The platforms must conform to the coordinate system	The platforms should appear on the screen when	As expected	**Fig E.13**

	correctly on the screen? I will visualise where I expect the platform to appear, then test if that is where the platform is actually drawn.		of the level, because the player has to interact with them (such as during collisions).	loading into a level.		
8	Do the levels get loaded correctly from a JSON file in secondary storage into a list in main memory.	A properly formatted JSON file with all the correct fields in each of the levels. (valid) A properly formatted JSON file without all the required fields. (boundary) An improperly formatted JSON file. (erroneous)	Because all objects in the level must be where specified in the JSON file. If there are any missing fields or invalid JSON in the JSON file, the level loading should stop, and an error should be thrown. This is to ensure that no error-prone or incomplete levels are loaded and played by the user.	Visible evidence for this would be that there are some levels presented on the level select screen, and clicking a button takes the user to a level.	As expected	**Fig E.14, Fig E.15**
9	Does the user jump correctly (only when on the ground)?	Pressing the jump input (spacebar) when on the ground. (valid) Pressing the spacebar when in the air (boundary)	The player must not be able to jump when the game should not allow it, as the levels would not be designed for this movement, and the player may be able to take unintended shortcuts to finish the level, making the game less engaging.	The player should jump when the spacebar is pressed and on the ground but not get an upwards impulse of acceleration when in the air.	As expected	**Fig E.16, Fig E.17**
10	Does the player move horizontally	Pressing left input and right input	The player must move in a predictable manner,	The player should hove left after pressend A and	As Expected	**Fig E.18** (at the start)**, Fig E.19**

	correctly?	and both at the same time.	so the user can get used to the player's movement more easily, making the game more engaging. (e.g. Not moving when the left and right input are both pressed.)	right after pressing D, and not move if both are pressed at the same time.		(moving left)**, Fig E.20 (**moving right)**, Fig E.21** (pressing both at the same time)
11	Are collisions detected and responded to correctly? i.e. Does the player get blocked by platforms, get killed by kill boxes when entering, win when colliding with the target, e.t.c.	Moving at a normal pace towards a platform. (valid) Moving really fast towards the platform. (boundary, to test no-clip resistance)	I must test that all interactions involving collisions work, to give the game as many working features as possible, to make it more engaging.	The player should be stopped from falling and moving horizontally when moving into a platform, should complete the level when hitting the target, and should be killed after hitting a kill-box.	As expected	**Fig E.22, Fig E.23, Fig E.24, Fig E.25, Fig E.26**
12	Is the player's movement frame-rate independent. (This includes both constant velocity and when the player is accelerating.)	Setting the game to 60 fps. (valid) Setting the game to 10 fps. (boundary)	The game should work on as wide a range of hardware as possible, so that the game is accessible to as many people as possible.	The player should have the same movement, whether playing on 30 FPS or 60 FPS	As expected	**Fig E.27** (jump height at 30 FPS), **Fig E.28** (jump height at 60 FPS)
13	Is an appropriate sound played when the player jumps?	n/a	The game should give appropriate feedback to the player when they take an action to make the game more responsive.	A bouncy sound should be played when the player pressed the spacebar while on the ground.	As Expected	n/a
14	Is an	n/a	The game should	A failure sound	Not	n/a

	appropriate sound played when the player dies?		give appropriate feedback to the player when something happens to the player to make the game more responsive.	should play when the player dies.	achieved because of time limitation	
15	Is an appropriate sound played when the player completes a level?	n/a	The game should give appropriate feedback to the player when they take an action to make the game more responsive.	A success fanfare should play when the player completes a level	As Expected	n/a
16	Is some promotional material shown (a special offer when the player completes a level and the Tekkerz logo when on the main menu screen)?	n/a	The game is made to promote the Tekkerz restaurant change, so must show promotional material effectively.	A special offer should be shown on the level completion screen and the Tekkerz logo should be shown on the main menu (what the player sees when they first load the game).	As expected	**Fig E.29, Fig E.30**
17	Is the game over screen shown when the user dies in a level?	n/a	The game should give appropriate feedback to the player when something happens to the player to make the game more responsive.	The game over screens should show when the player dies with a retry option and an exit option.	As expected	**Fig E.31**
18	Does all movement work correctly?	Pressing the wasd and spacebar keys.	Movement must be fully functional for the user to complete the levels.	The player should move when pressing the A and D keys and jump when pressing space and on the ground.	As expected	n/a
19	Does the user get a special offer after	Hitting the target and completing the level.	The stakeholder specified that the game must have sufficient	A special offer should show on the level complete screen	As expected	**Fig E.32**

	completing a level?		promotional material.			
20	Is all user progression saved correctly?	Making progress, saving and then exiting.	The user must be able to leave the game and return to where they left off to make the game more engaging.	When exiting the game and completing a level, the level select screens should show that that level is completed. The position and velocity of the player should also be saved so that the level can be resumed correctly when the player resumes later.	As expected	**Fig E.33** (before completing), **Fig E.34** (after completing a level), **Fig E.35** (the level loading the position and velocity of the player after they resume a level).
21	Can the user double jump?	Pressing the jump key while in the air.	This is an extra ability that will keep the user engaged.	When pressing jump for the first time in the air, the player should get a momentary boost of upwards acceleration like a jump.	Not achieved due to time constraints	n/a

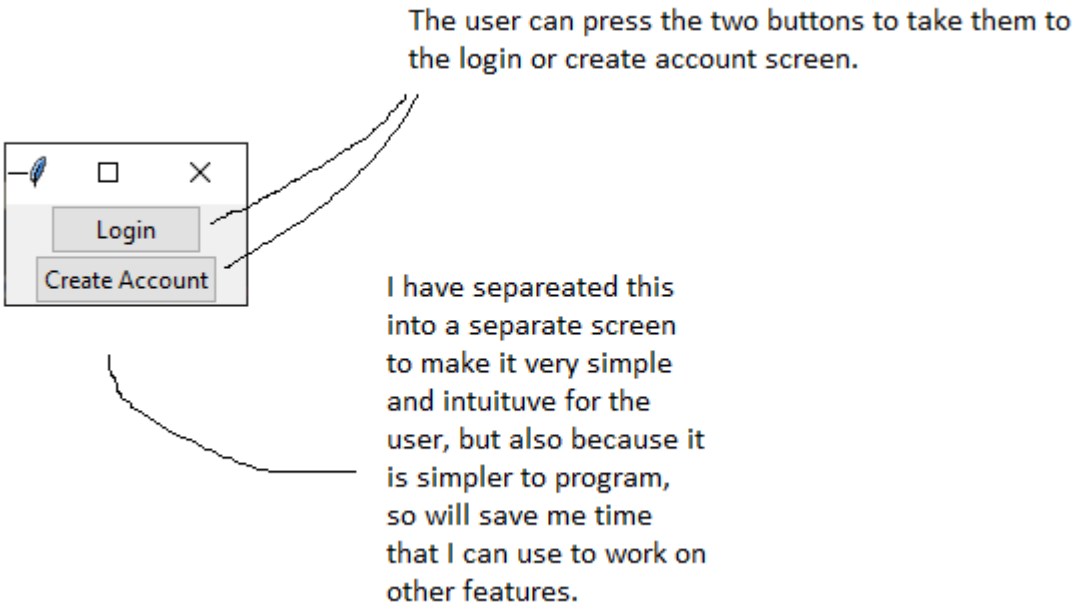
Robustness:

In this table I will describe, justify and provide evidence for tests used to analyse the robustness of the solution. I will do this to ensure that the features of my solution can be used flexibly and can be expanded on thoroughly in the future. Testing robustness will also mitigate against the solution causing the stakeholder issues with it breaking in the future.

Robustness Test #	Description	Test data (if applicable)	Justification	Expected outcome	Actual Outcome	Evidence
1	Entering a very long username and password into the create account screen.	"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa, etc" and "1111111111111111111111111111111111111111, etc"	Any length of username and password should be accepted and not have any errors when being saved to the users file in secondary storage.	The username and password should be accepted and saved to secondary storage	As expected	**Fig E.36**, **Fig E.37**

2	Moving very fast towards a very thin platform.	n/a	I want to test that the player can't clip through a platform if they move fast enough, making levels too easy or unplayable. The kill boxes and target use the same collision detection algorithm, so if this works, the collision with these other level elements is robust.	The player shouldn't clip through any platforms.	As expected	**Fig E.38** (moving very fast towards the platform), **Fig E.39**
3	Clicking a button and moving my mouse outside and releasing it, and clicking my mouse outside a button and then releasing the mouse inside the button.	n/a	The buttons should work intuitively (the same as buttons in similar games and user interfaces) so that the user can navigate the menu system easily.	Neither of these actions should cause the button to trigger.	As expected	**Fig E.40** (the mouse being released inside the button), **Fig E.41** (the mouse being released outside the button), **Fig E.42** (nothing happening due to these actions)
4	Having loads of platforms in a level.	The data used to make the platforms (too much to put here)	The game must support very large levels with lots of platforms because the game may be further expanded.	The level should load and run fine, maybe at a slightly lower frame rate because so many more collisions are being checked, but the platforms' collision is only O(n) so it should be fine.	As expected	**Fig E.43,** (the extra platform data) **Fig E.44** (there are more platforms not visible)

Black Box Testing, testing the usability

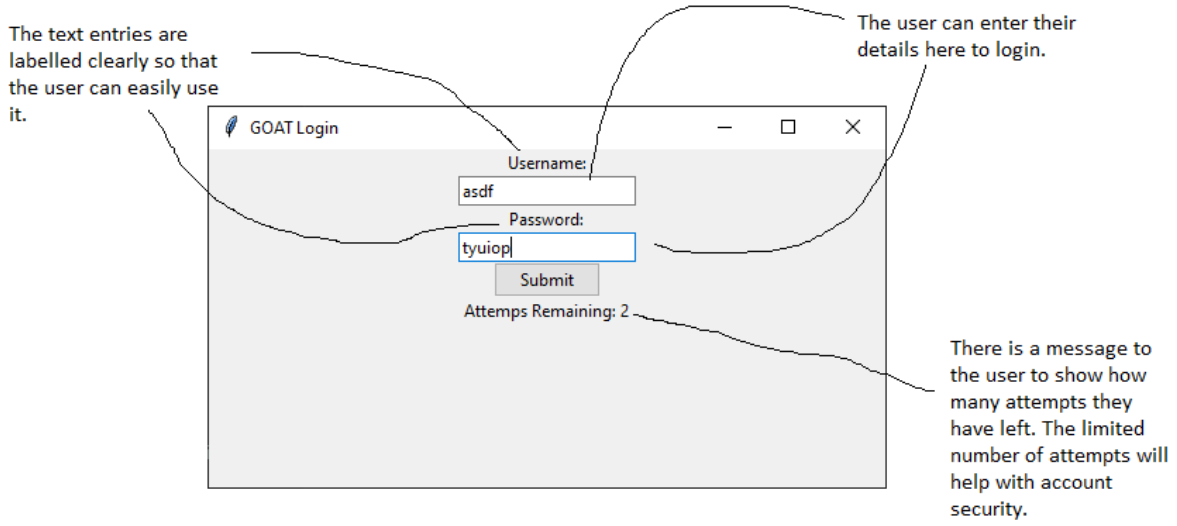


Me: This is a screen used to choose whether to create an account or login.

Stakeholder: I do not remember discussing this aspect of the proposed solution, but the buttons are very clear and readable. However, the buttons could be a different colour to the background to make them stand out.

Me: I made this screen very simplistic as it is just a screen that will show up momentarily before the user makes their decision and the background colour would be very easy to change as it uses python's tkinter which is beginner-friendly.

This is a partial success because this screen is fully functional, but its usability could be improved by changing the colours of the screen to be have more contrast and be more interesting and engaging for the target audience.



Me: This screen is a simple login screen that is clearly labelled that allows the user to access their account with their progression.

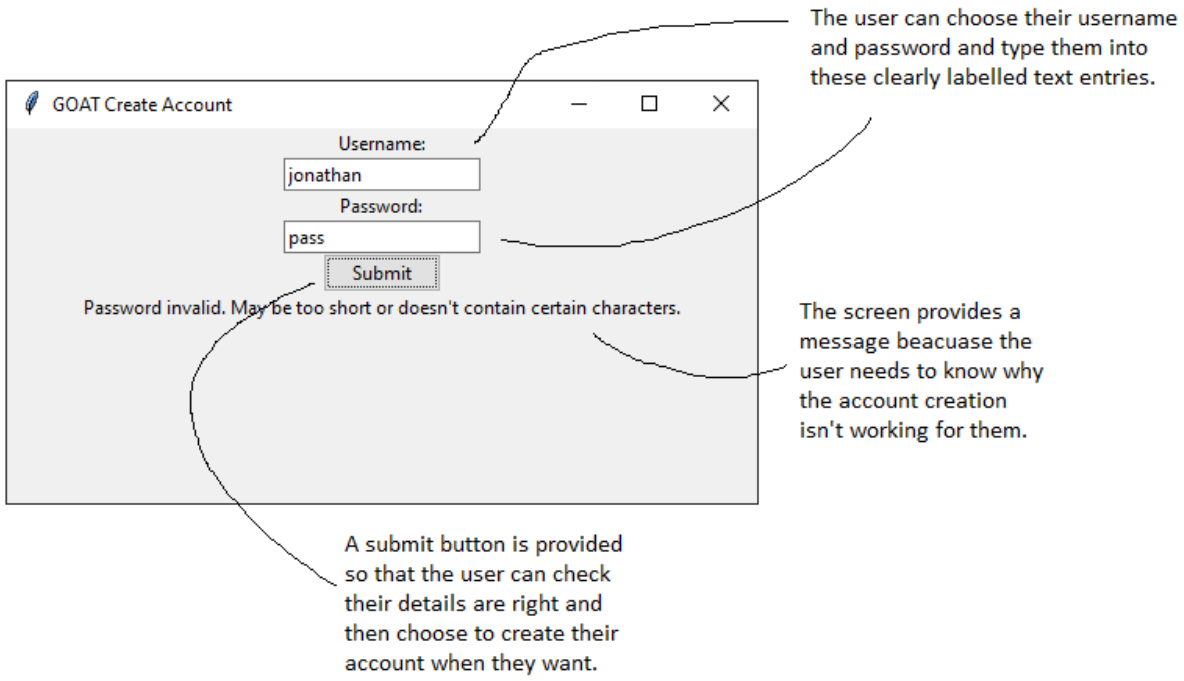
Stakeholder: As we discussed, this does look nice and simple for younger users. Providing security for accounts is also useful, but not critical as it is only a simple game to be used as promotional material and accounts should not contain any personal information.

Me: This system also allows for multiple users identified uniquely by usernames to use this game, so that the promotional material can be seen by as many people as possible.

Stakeholder: Good, although the screen could be more colourful and interesting.

Me: I had some time constraints and decided that this was a lower priority than producing a fully working game.

This is a partial success because the screen works well. However, this could be improved by changing the colours of the screen to give it higher contrast so that it is more accessible.



Me: This is a simple account creation screen so that the user can create an account and start saving their progression.

Stakeholder: This is a fine account creation screen. But, the message to the user could be more descriptive, considering they don’t know what the requirements are for the password.

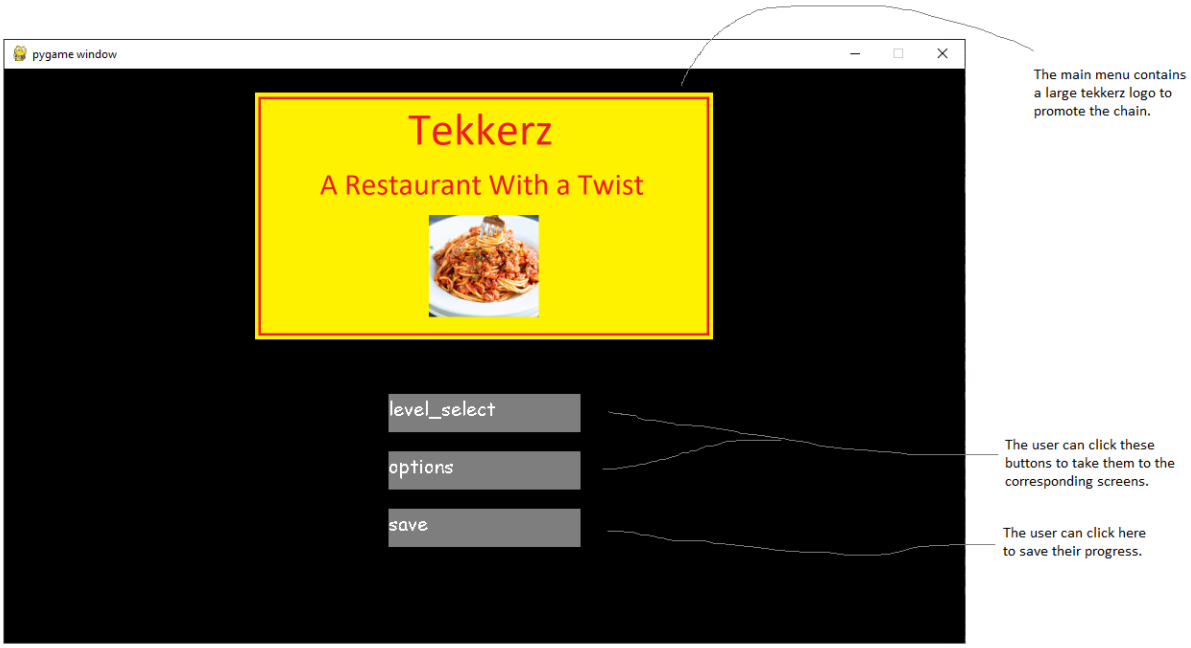
Me: Yes, the message could be more descriptive, but I have written the code in a way that the password requirements can be changed easily in the future. (At the moment, the only requirement is that it is five characters or more.)

Stakeholder: What about if the username or password contains spaces?

Me: There are two other messages to the user for these cases. “Username cannot contain a space character.” and “Password cannot contain a space character.”

Stakeholder: That seems clear.

This is a partial success because the screen is mostly fully functional, with the user being able to create an account easily. But this screen could be improved by giving the user a more descriptive message to communicate the reason that an account creation failed. For example, a more specific reason why the password was not suitable.



Me: This is the main menu, containing necessary navigation and a large Tekkerz logo, as you suggested.

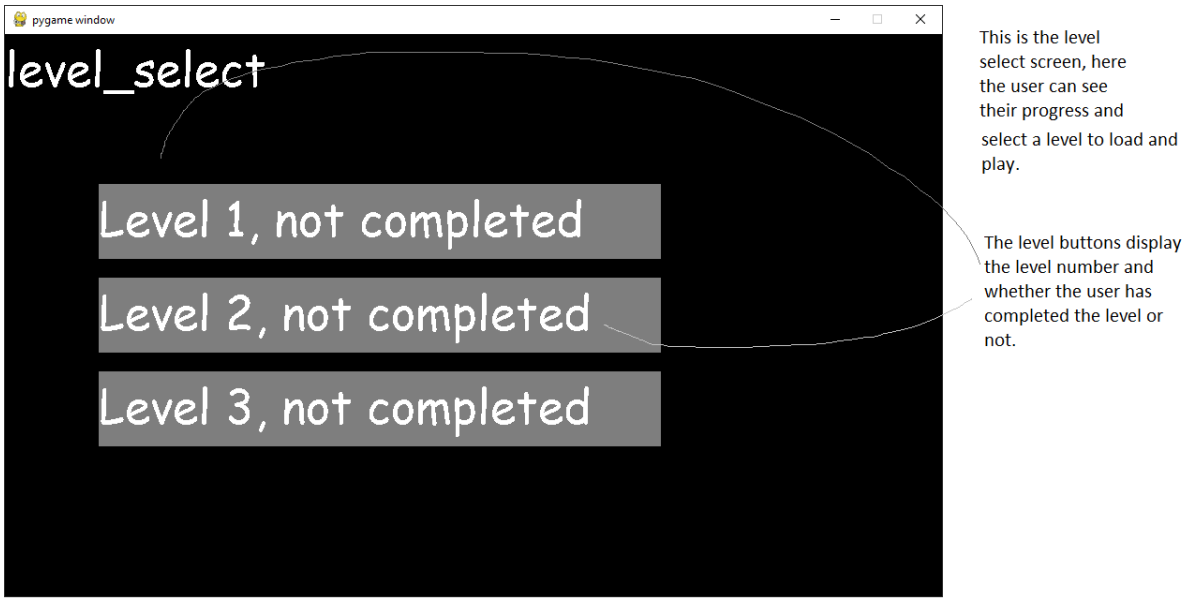
Stakeholder: The Tekkerz logo is certainly big enough and the buttons have a good high contrast with the background. However, the buttons could be more interactive [after seeing them function]. For example, highlighting when the user hovers over them to make it more obvious that they can be clicked and make their use more intuitive.

Me: Yes, I could have done that, and it would make the buttons' use more intuitive, but it would be quite a lot of work for a fairly minor usability improvement.

Stakeholder: There could also be a button for quitting.

Me: The user can quit by pressing the cross.

This is a partial success because all the elements of this screen are fully functional with adequate promotional material due to the request of the stakeholder. However, I could have made the buttons more interactive by highlighting them when hovered over, to make the user aware that they can be clicked.



Me: This is a simple level select screen. It shows the user that they are on this screen and gives them a list of levels to choose from.

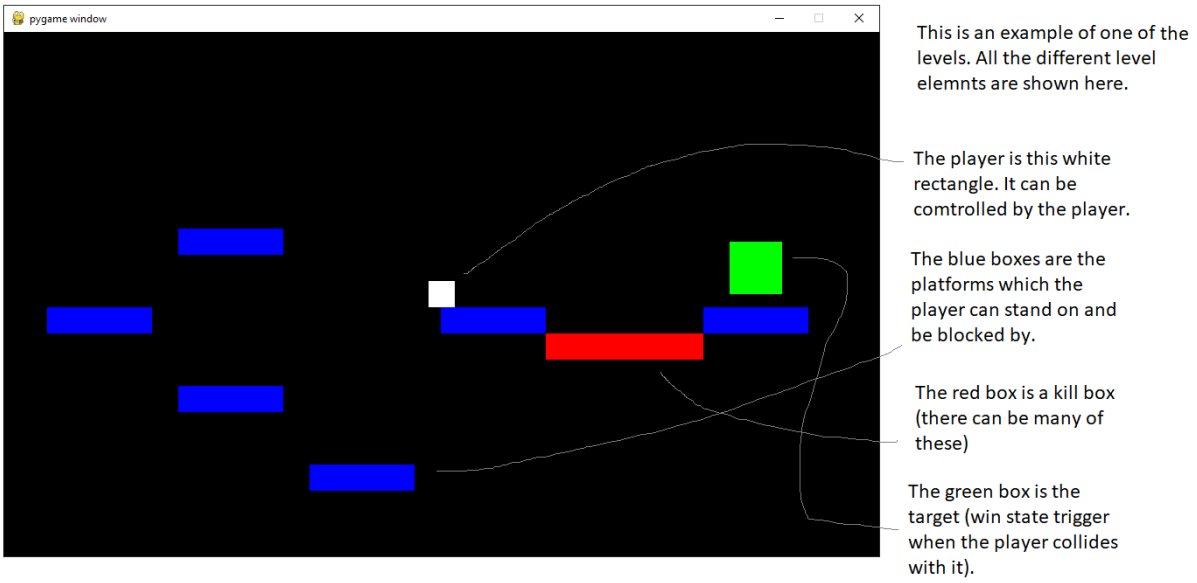
Stakeholder: This screen looks very functional and easy to use, but could be made more colourful to keep the attention of the younger audience. The completed element of the buttons could be more interesting for example a graphic like a tick.

Me: The text on the button was already implemented and to do a tick, I would have to create an image and position it correctly for every button. This would be time consuming and possibly not very robust when changing the locations of the buttons.

Stakeholder: Also, there are only three levels, I asked for many more than that.

Me: The levels take a lot of time to create, but the number of levels is easily expandable because they are stored in a JSON file format which is human-readable.

This is a partial success because all the buttons work, and take the user to the desired level, but the buttons could have been made more interactive (as described previously). Also, I should have changed this screen to more faithfully replicate the layout in the design section, with ticks for when the user has completed a level.



Me: This is an example level to show all the different level elements. It has horizontal movement, jumping and kill boxes as I discussed with an example member of the target audience.

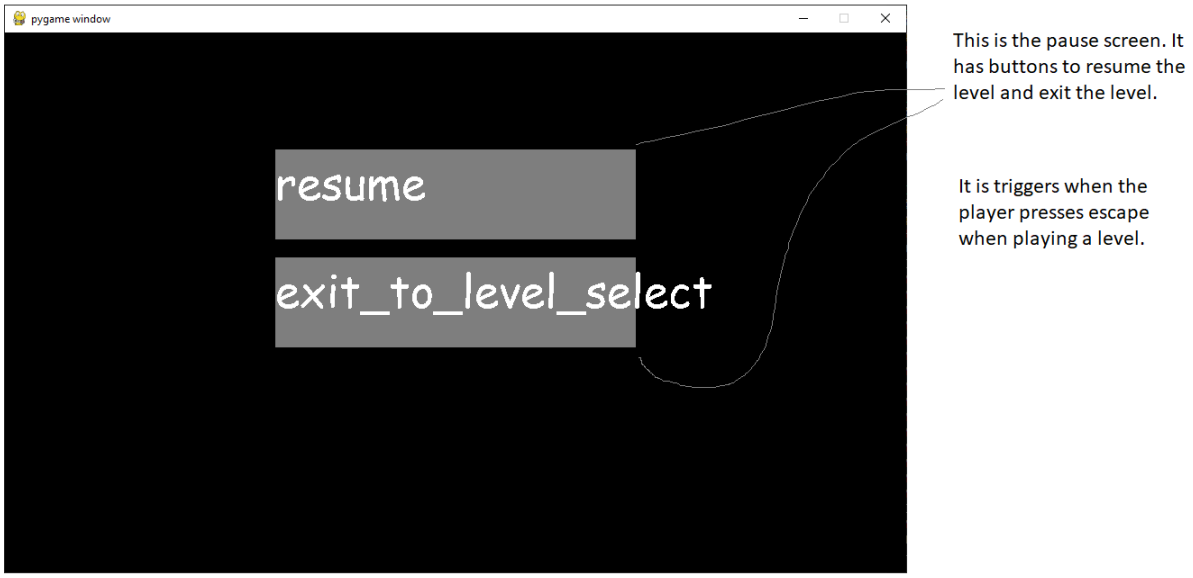
Stakeholder: This is good. It has bright colours to engage the younger audience and has many interesting level elements. Though double jumping could be added to make the gameplay more engaging and to increase the complexity ceiling of the levels.

Me: This would be ideal, but I was not able to implement this due to time constraints.

Stakeholder: The level also seems quite simple which is good for the younger audience. But, it would be nice to have sprites.

Me: This would take a long time, and the blocky colours add to the retro aesthetic.

This section is a success with all required aspects of the game fully working. However some optional features were not implemented due to time constraints.

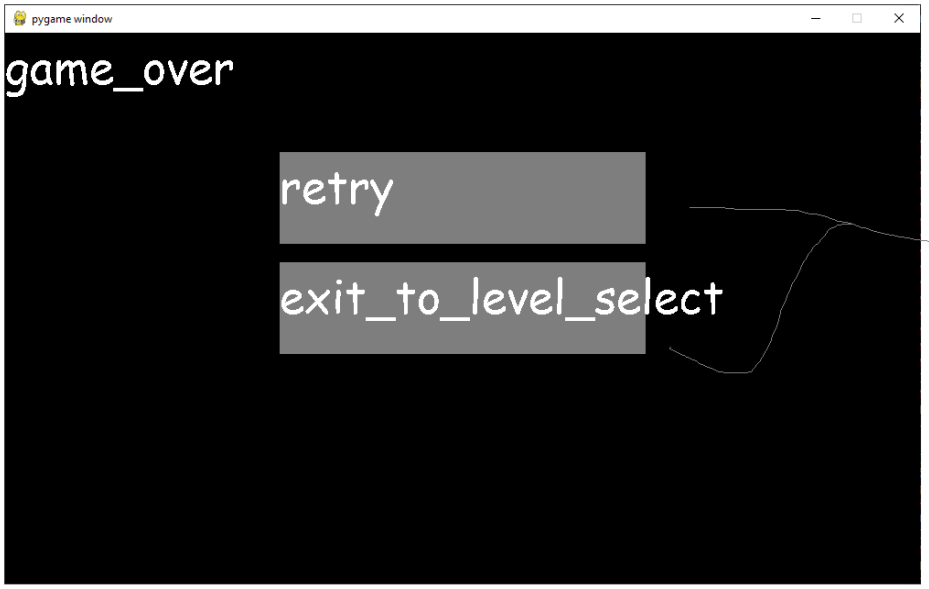


Me: This is the pause menu used by the user to navigate out of a level.

Stakeholder: This is good in that it is simple and intuitive, but it could be polished more, like by making the text fit inside the button. Also, I would expect a save button to be here so that the user can save their progress in the level.

Me: The save button is on the main menu and a save prompt appears when exiting the game. Progress in a level is automatically saved. I also didn't have time to polish some of the UI screens due to time constraints.

This screen is a success because all the necessary features are implemented. Some features originally intended for this screen have been moved elsewhere.



This is the game over screen - what the user sees when they fail a level.

These are the buttons for quitting and restarting.

Me: This is the screen to alert the user that they have failed a level and gives them the options to retry and exit.

Stakeholder: This is a good game over screen, though the game over graphic could be improved and the UI could be more polished.

This is a partial success, as all the planned functionality is available to the user. However, the game over graphic should be more interesting to engage the user, and a sound could be played on this screen (as the player has just died).

The level complete screen with a Tekkerz special offer as a reward.

Options for haveing another go at the level or exiting.

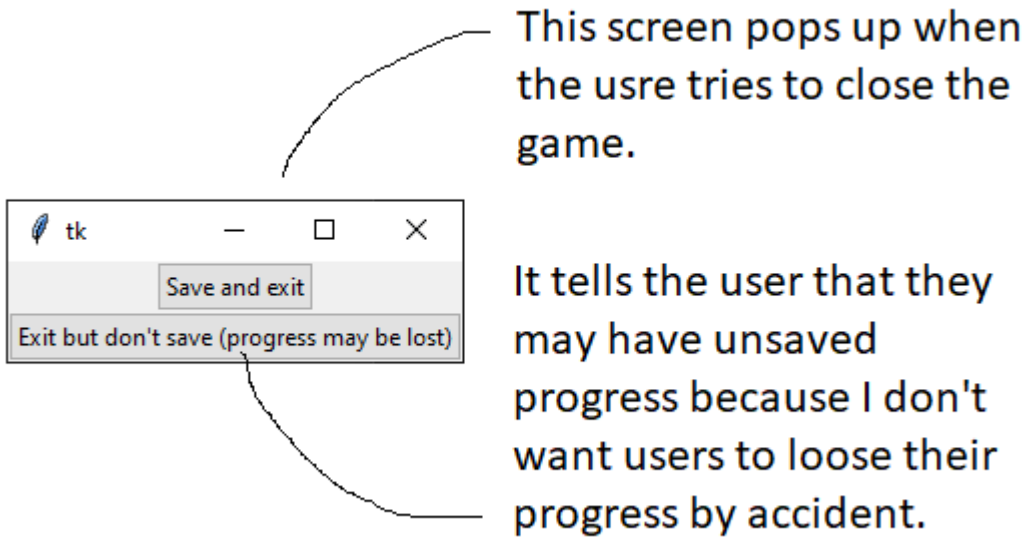
A winning fanfare is also played.

Me: This screen contains the special offer as discussed, and options for user navigation.

Stakeholder: The special offer is good. It is very noticeable and bright. [after the sound effect plays] The fanfare is also a nice addition to reward the user. However, the level complete graphic could be more interesting and colourful.

Me: I did not have time to create some graphics.

This is a partial success because all the buttons work on this screen. However, I could improve the usability of this screen by making the level complete graphic more interesting, to make the purpose of the screen clearer to the user.



Me: This screen will show up to notify the user that they should save their progress. It is useful because a user may become frustrated with the game if they lose their progress by accident. It also gives a backup to the slightly unorthodox approach to saving progress that I have taken.

Stakeholder: I do not remember discussing this, but it seems like a useful addition with clear options.

This is a success because it acts as a helpful prompt to the user to save their game progress. This is an extra feature that I did not originally plan to implement.

Conclusion from blackbox testing:

Generally, I think the feedback from the stakeholder was positive as I have implemented all the main features asked for in the solution. He seemed especially content with the promotional material and the gameplay. However, he seemed less happy with the user interface as (although it worked) it was not very polished and should have been better designed with more colourful graphics.

If I were to make some further improvements, I would add more levels to keep the user engaged for longer. I would also change the buttons to be more interactive, for example highlighting when the user hovers over them and fitting the text into the buttons. Furthermore, I would add some elements to the movement during gameplay, such as double jumping, dashing and wall-jumping/climbing.

Success of solution

How the solution as met the success criteria set out in the analysis section

Here, I will assess how well the success criteria have been met, and provide evidence for their success or failure. I will do this because it is important to measure the success or failure of my solution and to work out how I can improve the solution in the future. This will allow the stakeholder to more accurately measure the success of the solution.

1 - Runs at a suitable frame rate:

I have met this success criterion well by making the game and collisions simple, and limiting the use of resource-expensive functions such as the square root. I have also made the game run correctly at any framerate (this is dependent on the hardware) by making the player's movement dependent on the time between frames. I have done this because the game should run on as wide a variety of hardware as possible, so that the promotional material is accessible to as many people as possible. Evidence for the stable 60FPS frame rate can be seen in the video, but the frame-rate independent movement can be seen in the white-box testing for functionality test 12 and at **Fig E.27** and **E.28**.

2 - User friendly user interface:

I have partially met this success criterion by providing a user interface with simple text entries and buttons using the python tkinter library for user interfaces (which can be seen in the white-box functionality test 1 and 2 and in **Fig E.1, Fig E.2, Fig E.3, Fig E.4**), which are used for the create account and login screens because this is a simple way to create a UI and it doesn't necessarily have to be part of the same screen as the gameplay (which I used pygame for). However, the user interface in the menus of the game was lacking because I had to create a button class for use in pygame from scratch. This also meant that I didn't have time to make buttons highlight when hovering (like in tkinter) and sometimes the text used for a button didn't properly fit inside it. This took away from the overall polish of the game.

3 - Feature rich user interface:

I have fully met this success criterion by creating all the necessary parts of the user interface:

- A main menu
- A level select screen with level numbers
- A text entry input
- Buttons
- A pause menu

All these are fully functional. This allows the user to easily navigate the game. To achieve this, I have created a button class and used the python function call stack to store the order of screens that the user has entered. Evidence for this can be found in tests 1, 2 and 16 in the white-box functionality testing and in **Fig E.1,2,3,4,11,14**.

4 - The user must be able to simply create and log into an account

I have fully met the requirements of this success criterion. I have achieved this by creating login and account creation screens using python tkinter. They contain intuitive text entries and submit buttons. Where necessary, they display messages to the user explaining information such as why usernames and passwords aren't accepted and how many attempts the user has left to log in. All the data the user enters is stored in a file in secondary storage, which stores all information about user accounts. Evidence for this can be found in whitebox functionality tests 3 and 5 and in **Fig E.1,2,3,4,5,6,7**.

5 - There must be sufficient promotional material for Tekkerz in the game

I have fully met this success criterion to the extent that the stakeholder suggested. There is a Tekkerz logo on the main menu and the user gets a special offer for Tekkerz when they complete a level. Evidence that the stakeholder is content with this can be found in the black-box testing discussions with the stakeholder while testing the main menu and the level complete screen. Evidence of the features themselves can also be found here.

6 - The game must contain enough playable content to keep the user engaged for a reasonable amount of time

I have partially met this success criterion by creating the gameplay levels. I have created three levels when the original plan was to create ten. This is because the time constraints of the solution meant that I couldn't create many levels, as it is very time consuming. However, the levels are easy to add to as their data is stored in a JSON file which is human-readable and editable data. The stakeholder wanted many more levels, even more than I originally planned, but if I had more time, I may have been able to fulfil their request.

7 - The game must be visually appealing with simple retro graphics

I have partially met this criteria, because I have created simple graphics for the game, like for the promotional material, but the player and other level objects are just solid blocks of colour. This element of the design is not critical for the solution, but I would have added more graphics if I had more time. However, the current graphics are still quite effective because they are very colourful, which will interest the younger audience, and it is easy to differentiate between the different objects within the level.

8 - The game's skill ceiling must be high

I have met this criteria quite well. This is because the levels are quite difficult. I have measured this by allowing an example member of the target audience to play the game, who had some difficulty completing the levels. However, by adding more levels, the skill ceiling could be increased to increase player interaction. Also, if I added more levels, I could make the beginning levels easier so that beginner users won't get frustrated by the game. This is a difficult criterion to provide evidence for.

Me: How do you like these levels?

Target Audience Member: They are good, but quite difficult , and I had some trouble completing them.

9 - The game's movement must be responsive and versatile

I have met this success criterion by making the jumping and movement responsive. I have achieved this by making the jump fast and quite high, and making the movement on the ground feel quick and sharp compared to the player's movement in the air. This makes the movement feel responsive because of the juxtaposition between the movement in the air and on the ground. I have made the movement versatile by allowing the user to navigate the levels quickly and smoothly. If I had more time, I would make the movement more versatile by adding double jumping, dashing and wall climbing/jumping. However, I could not achieve this due to time constraints.

10 - Information about the game's state must be shown to the user in a clear way

I have fully met this success criterion by implementing a level complete and game over screen (evidence shown in white-box tests 17 and 19 and in **Fig E.31** and **Fig E.32**). This presents the game's state in a clear and intuitive way to the user. I have also shown the state of user progression on the level select screen (shown in **Fig E.12**). This fully meets the criterion set, however, if I were to improve on this system, I would show the user a measure of how far they have progressed in each level on the level select screen.

11 - Each level must include a way to win and a way to fail

I have fully met this success criterion by implementing a target which the user is aiming for (**Fig E.25** and **Fig E.26**). This will update the user's progression and take them to a level complete screen which is the user winning. Each level will also contain some kill boxes which provide a way for the user to fail a level. When the player collides with these kill boxes, the user will be taken to the game over screen (evidence for this can be seen in test 11 in white-box functionality testing **Fig E.23** and **Fig E.24**), from which they can retry or quit. This is the user failing. If the player goes out of bounds, they also fail. I believe this meets the criterion adequately because it provides a way to access the win and fail states. The out of bounds rule also allows the player to retry even if they fall into nothingness and don't hit a kill box.

12 - A user must be able to save the state of the game to secondary storage

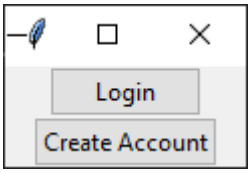
I have fully met this success criterion by providing the user a save button on the main menu. As the game is played, progress is saved to the user's progression dictionary. This is then saved to a file in secondary storage containing all the users' progression when the user clicks the save button on the main menu. The user also has the option to save when they exit the game, when a tkinter screen will pop up and remind the user to save. Evidence for this can be seen in white-box functionality test 20 and in **Fig E.33,34,35**.

13 - Actions in the game must give appropriate feedback to the player

I have partially met the requirements of this success criterion. A jump sound is played when the user jumps to make the game's movement feel more responsive. The player's movement is also clearly visible (**Fig E.19,20**). Another form of feedback is the graphics and sound presented when the user completes or fails a level. This applies to feedback to the user because it shows the user that the actions they took caused a change in game state. The level complete sound is also a fanfare which suggests victory. If I had more time, I would have added a sound when dying to give more responsive feedback to the user.

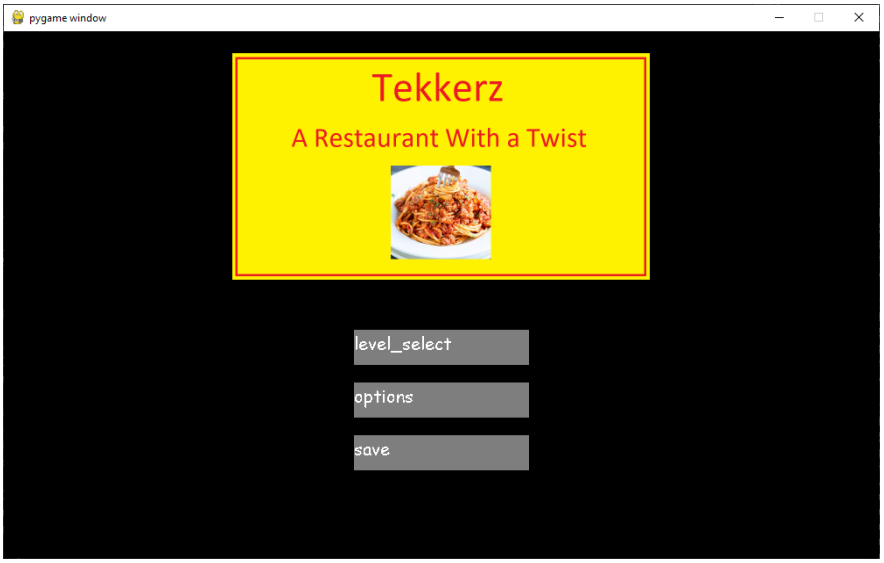
Evaluation of the User Interface Design and Usability Features (from the design section)

Login or create account screen:



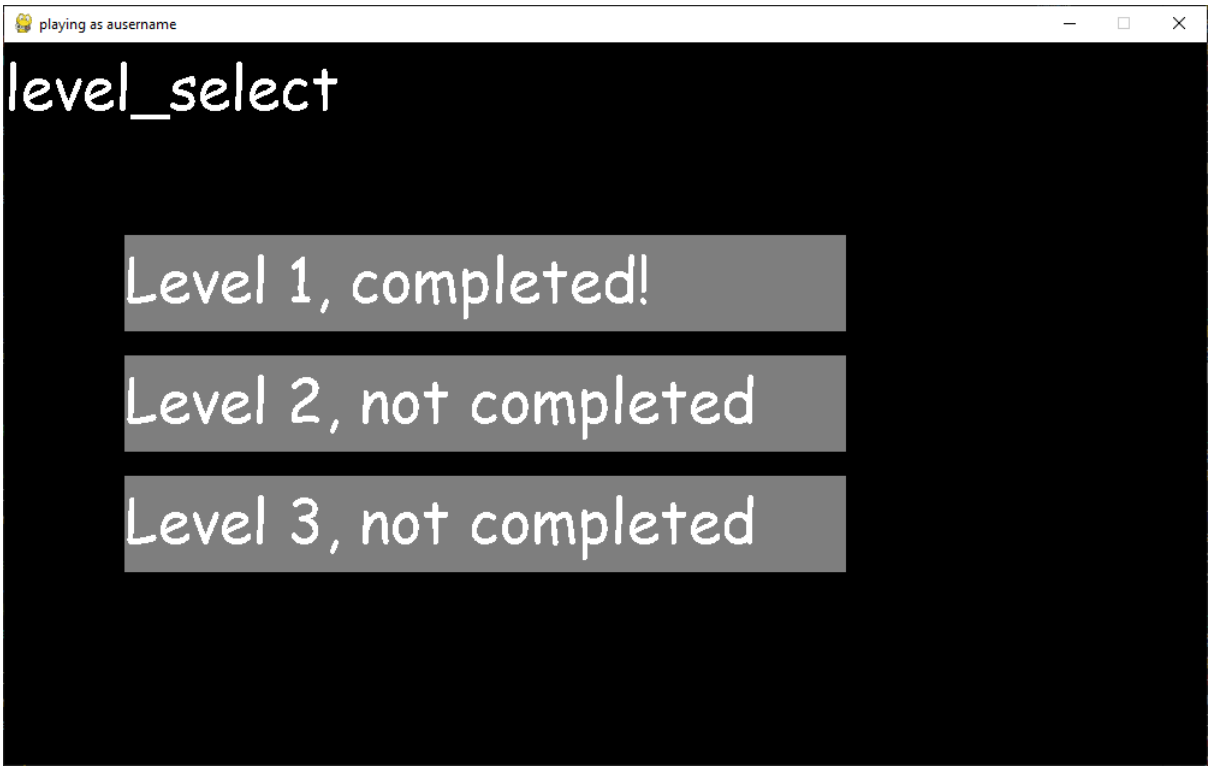
This is a very simple UI screen. This has met the success criteria set in the design section by placing the Login button above the create account button. The screen is also very simple and intuitive so is easy to use for the user. The colours are also the same as was set out in the design section. If I had more time, I would resize the window to make the buttons bigger, and maybe change the colours. Evidence for testing this usability feature can be found in the first discussion with the stakeholder in the black box testing section.

Main Menu:



This is the main menu. The screen that the user will first see after logging. I have mostly met the requirements for this screen set out in the design section. The logo is very large to provide maximum promotional material and the buttons are in the correct colours. All the buttons are in the right order, for example the level select button is at the top as it should be the most commonly used button. However, there is no exit button as the user can just click the cross in the top right. I have added the save button because I slightly changed the way I implemented the progression saving system during development. Evidence for the testing of the usability of this feature can be found in white box test 16 and the black box testing with discussion with the stakeholder about the main menu.

Level select screen:

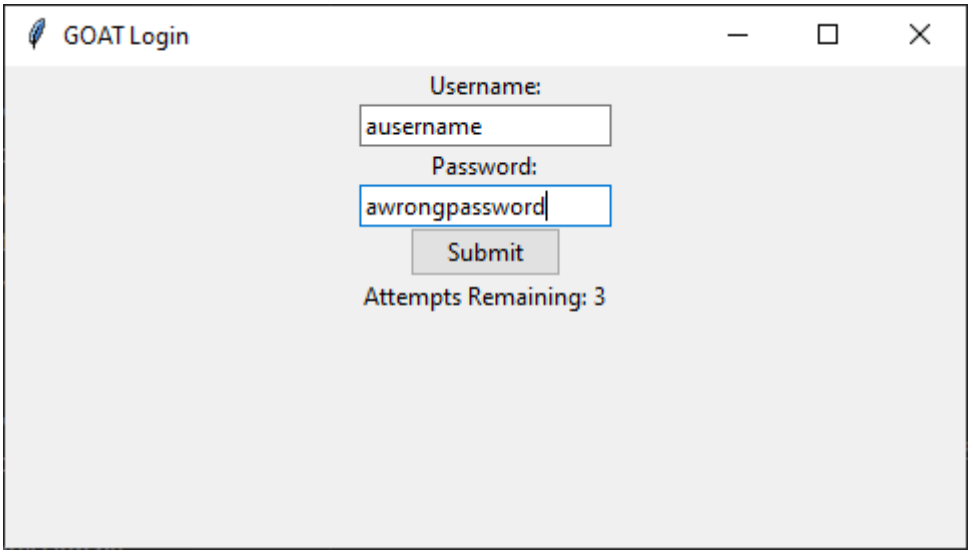


I have partially met the criteria for this screen in the design section. The screen is clearly labelled, and there are fully functioning buttons which include information about the user's progression. However, the buttons are not displayed in a grid because displaying them in a vertical column was easier. This way that the user progression is displayed is also different (text instead of a tick graphic) because I didn't have time to create the graphic and change the code to display it or add it to the button class. But, the screen is still intuitive and easy to use. Evidence for testing the usability feature can be found in the black box testing of the level select screen along with the discussion with the stakeholder and test 20 in the white box testing section.

Options Screen:

I have not met the requirements for this section because the screen is empty. This would have taken a long time to create and would contain relatively unimportant functionality, so I decided to spend my time elsewhere, developing other features.

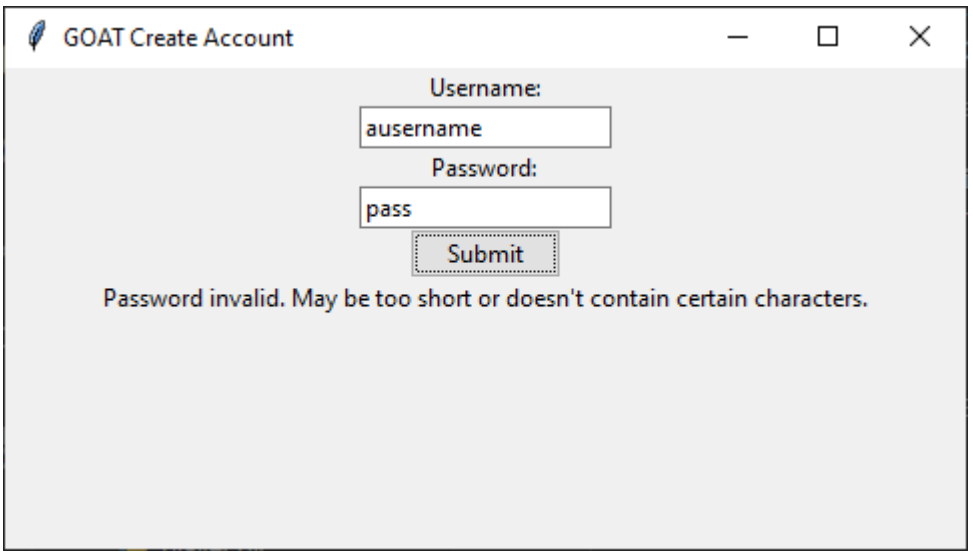
Login screen:



This is the login screen. It is simple and intuitive and provides the user a message to tell them how many attempts they have remaining. Evidence for the testing of this screen can be seen in the black box testing for the login screen and the white box functionality test 2. If I

had more time, I would resize the interface elements and make the password not visible. I have fully met the criteria for creating this screen from the user interface design section.

Create account screen:



This is the create account screen, which contains text entries used to enter details to create an account. I have fully met the success criteria for this screen. All the elements are the right colour, so as to be clearly visible and easy to use. I've added a message to the screen which will show the user some error messages as shown here. Evidence for this screen being tested can be found in the black box testing and discussion with the stakeholder for the create account screen and in tests 2 and 3 in the white box testing for functionality.

Maintenance and Future Development

Maintenance and maintainability of the solution:

Maintainability is a measure of how easy a program is to be maintained in the future, this includes how modular the program is i.e. splitting the program into functions, procedures and classes to make expanding the program beyond its current functionality easier. This is important because in the future, new features may need to be added to my solution such as new movement features like double jumping, dashing and wall climbing/jumping, or new interface features such as an options menu.

To make my program more maintainable, I have separated all the elements of a level into separate classes. This approach naturally applied to game objects as they appear as physical things on the screen. My tkinter UI screens are implemented as classes which inherit from the tkinter screen class, which means more functionality can be added in the future via methods. Furthermore, the in-game screens I used are all represented by functions, so that I can use the in-built python function stack to store the different screens the user has accessed, allowing the program to easily switch between screens.

I have also used comments to describe how sections of code work and why they are implemented in a certain way. This is because it will help someone else understand my code, so that they can add more functionality to it in the future. Descriptive variable names and consistent naming conventions have also been used to make my code more readable.

If I had more time, I may have switched to an object oriented approach to the in-game menu screens to store and change their state more easily.

Future Development:

One feature that would improve the system in future development would be the addition of an options screen to allow the user to choose a framerate because the game may use too much of their computer’s resources, and a way to change the resolution of the game screen to accommodate for different hardware. I would also add more promotional material to the game to improve the results for the stakeholder.

Another feature I would add is more versatile movement. This would include double jumping, dashing and wall jumping/climbing. This would allow the levels to be more versatile and interesting, so as to keep the user engaged with the promotional material for longer. By allowing the player to unlock these abilities as they progress through the levels, the user could feel a greater sense of achievement because they have more elements of gameplay to measure their progress against, rather than just which levels they have completed and the skills they have learnt to complete those levels.

Finally, I would add a lot more levels. This is because it would increase the user engagement and give the user more to play, and more special offers to receive. However, I did not have time to implement these features due to time constraints.

The program could be developed to deal with usability limitations by adding some more accessibility options such as a screen reader for the menus. The menus could also be made more colourful and interesting, as suggested by the stakeholder, to make the game more engaging for the younger audience.

I would add double jumping by having an attribute in the player class that stores how many times the player has jumped while in the air and then checking this variable is lower than 1 when the player attempts to jump while not grounded.

I would implement dashing by having the player gain momentary acceleration when the “F” key pressed in the direction that the player is trying to move (i.e. left if “A” is pressed and right if “D” is pressed). This would have a cooldown timer associated with it which would only allow the player to dash if the timer is 0, and be set to 5 seconds when the player dashes.

I would implement wall sliding by having an upwards acceleration applied to the player that is proportional to their downwards speed when the player is in contact with the side of a platform. Then, to add wall jumping, I would apply an acceleration to the player away from the wall and upwards when the jump key (spacebar) is pressed when the player is sliding down the wall.

To make these abilities related to progression, I would have attributes on the user class which determine whether the player has each of these abilities. Then, I would check these values when the player tries to double jump, wall jump, wall slide and dash. I would set one of these values to true at the end of each level.

To add an options screen, I would follow the design of the options screen user interface in the design section. I would position lots of buttons and text on the screen to allow the user to select options such as screen resolution in an intuitive way.

I would add more levels by adding more entries to the array in the “levels.json” file. These would get progressively larger and more difficult, considering the new abilities the user gets.

Final Conclusion:

Overall, I think my project has been a success. Most parts of my plan have been fully met, along with most of the success criteria set out in the analysis and design sections.

Of the parts which I have not fully met (due to time constraints), I designed related sections of code such that they can easily be expanded upon to implement these extra features. For example, I have made my design as modular as possible, so that individual sections (such as the different in-game menus) can be added to without having unintended side effects on the entire program. I have also added comments to my code to explain how my code works to future developers and to explain to them why I have implemented certain features in the

way that I have. Proper indentation and consistent variable naming conventions have also been used to make my code more readable.

Regarding the parts of my design that I have implemented, I have analysed the problem with discussions with the stakeholder, and tried to deliver a product as close to the brief as possible, within the limits of the time I had to create the solution. For example, I have delivered on creating engaging levels which contain all the necessary features to keep the user interested and allow them to complete each level, however the number of levels in my game is limited due to time constraints. Another part of the game which I fully delivered on is the accounts, users, login and progression system. I achieved this using object-oriented programming. By using objects to store the user data and a dictionary containing level progression objects in the user object to store progression data.

The stakeholder’s response to my solution was positive overall, because I implemented the promotional material for his restaurant well, with an easy to use user interface which is ideal for the younger target audience. However, the one part of the game he was most concerned about was the number of levels, however these can be easily expanded upon by editing the levels JSON file.

In conclusion, I think my solution went well, with most success criteria fulfilled, and with lacking areas ready to be easily expanded if the time constraints are lifted.

Evaluation Figures

Fig E.1

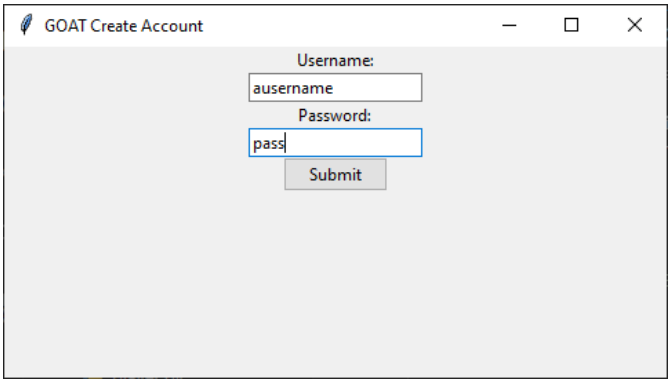


Fig E.2

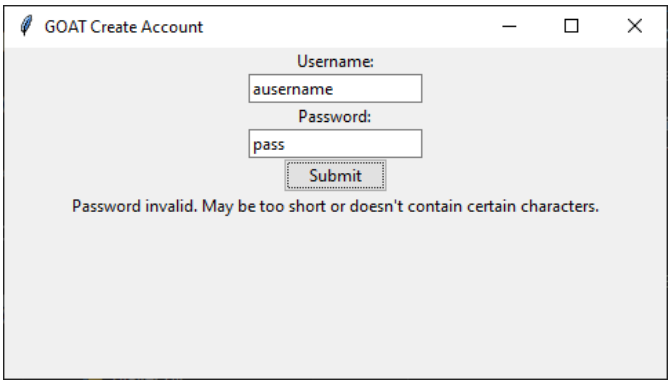


Fig E.3

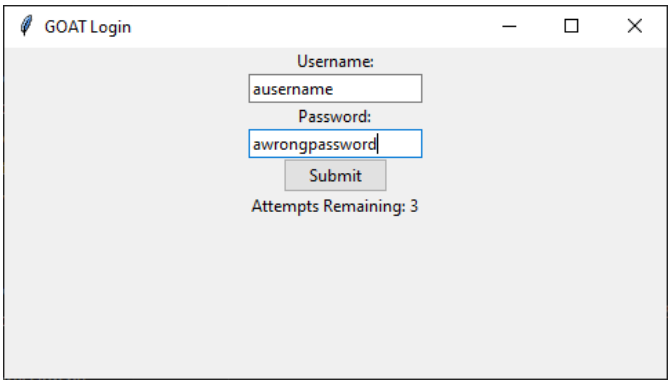


Fig E.4

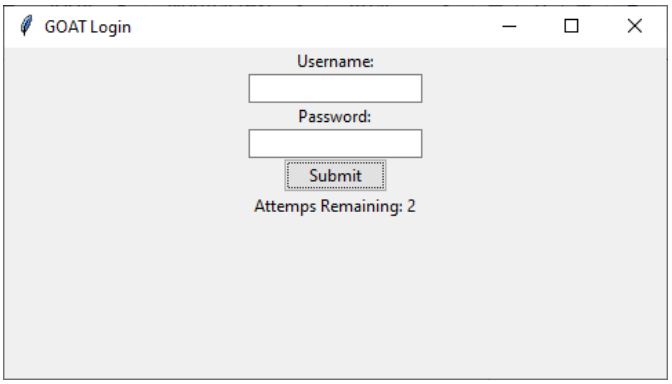


Fig E.5

```
��m������]��
users.user��User���)��}�(�username��
my_username��password��very_secure_password��
progression�}�uba.
```

Fig E.6

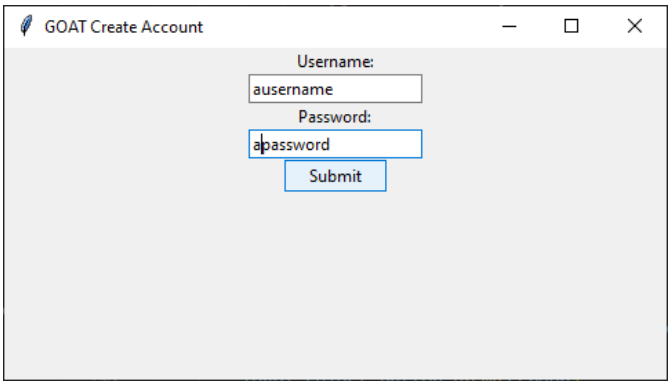


Fig E.7

```
��������]�(�
users.user��User���)��}�(�username��
my_username��password��very_secure_password��
progression�}�ubh)��}�(h�  ausername�h�   apassword�h
}�ube.
```

Fig E.8

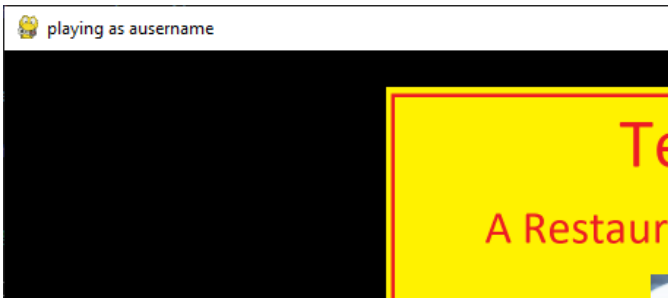


Fig E.9

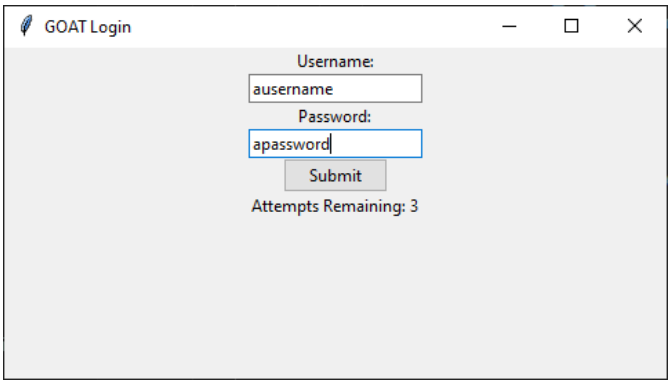


Fig E.10

Fig E.11

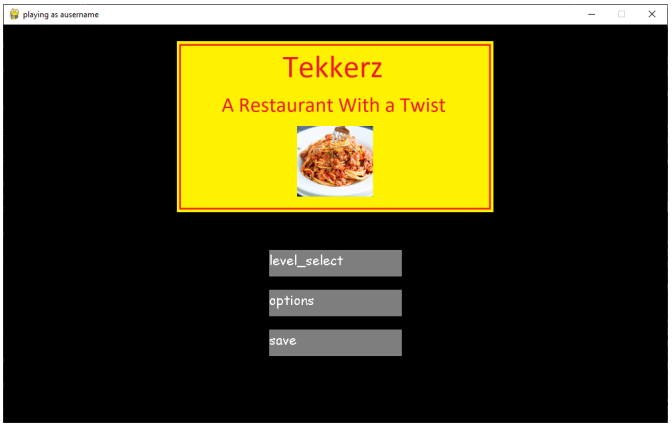


Fig E.12

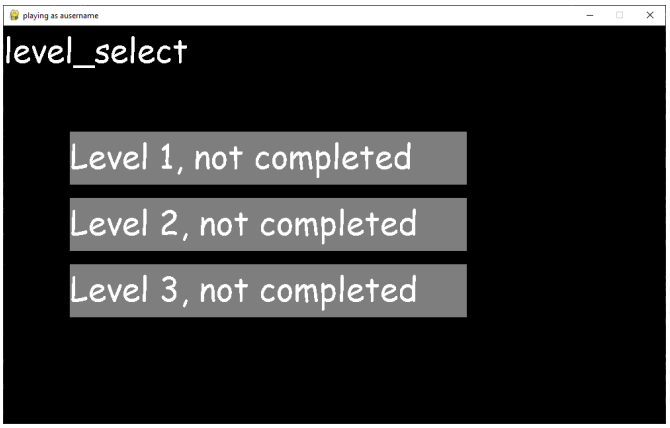


Fig E.13

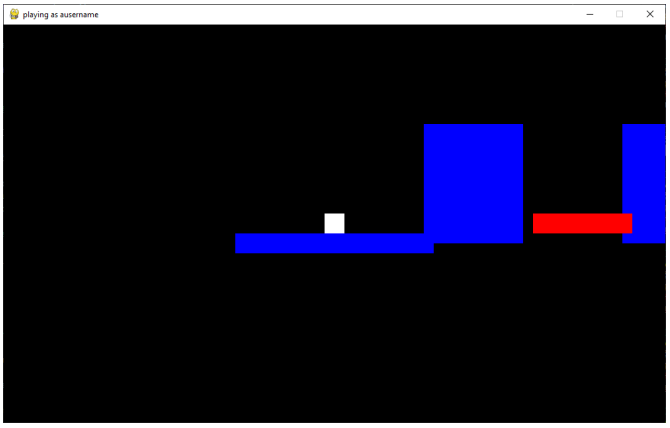


Fig E.14

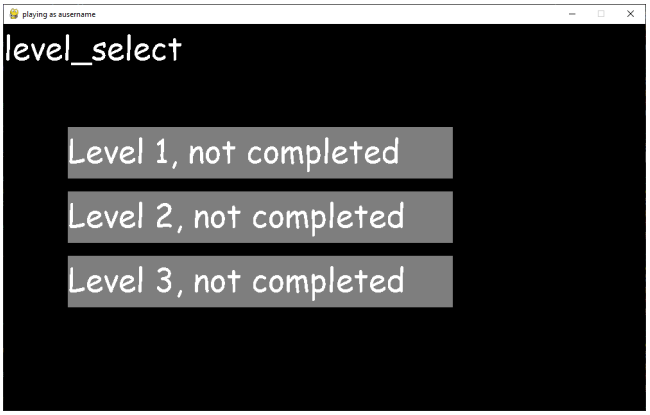


Fig E.15

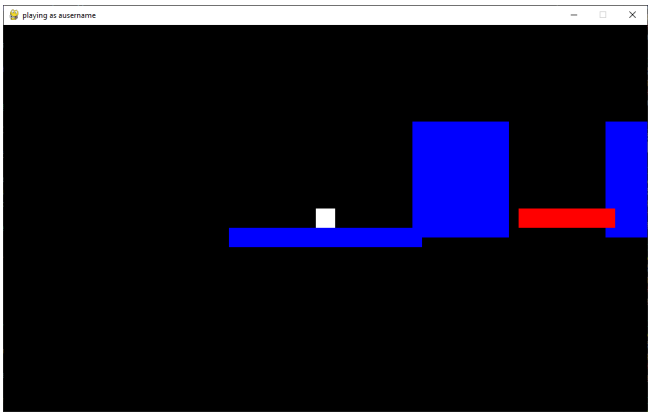


Fig E.16

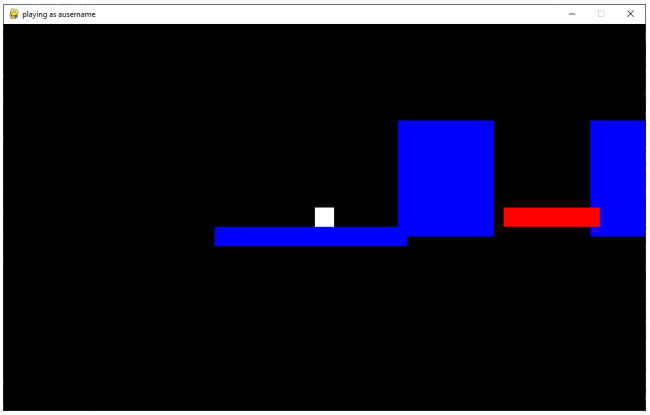


Fig E.17

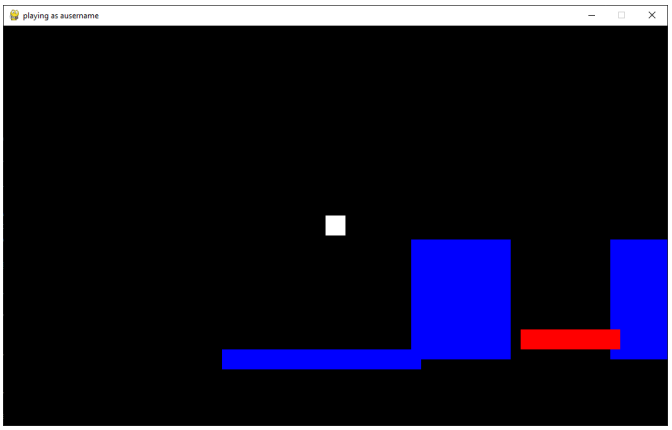


Fig E.18

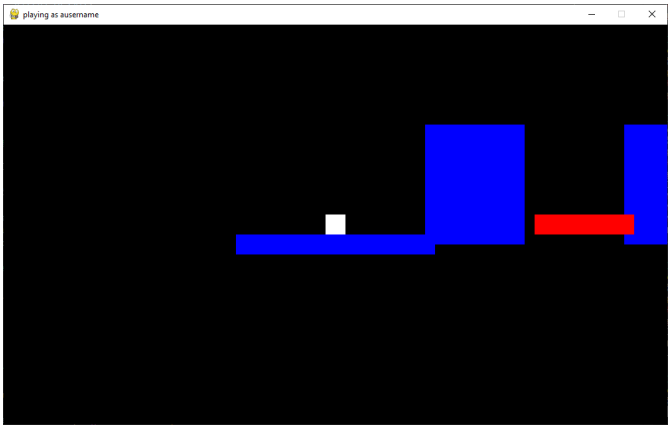


Fig E.19

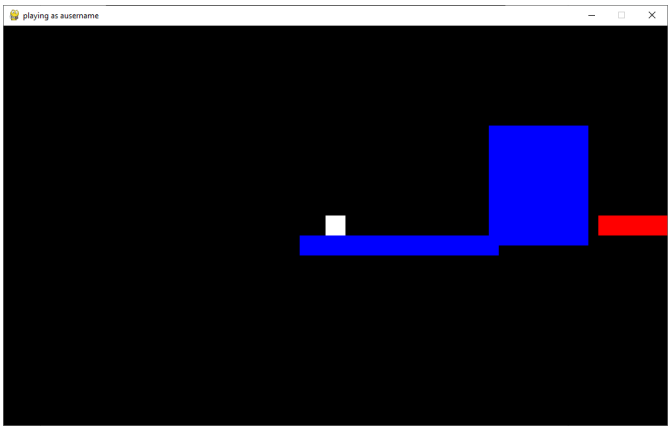


Fig E.20

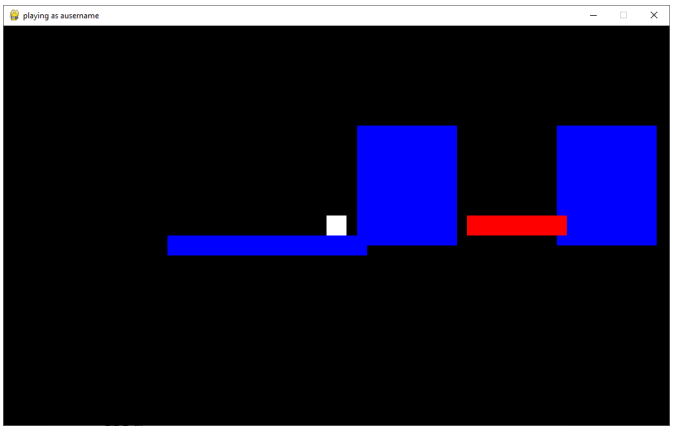


Fig E.21

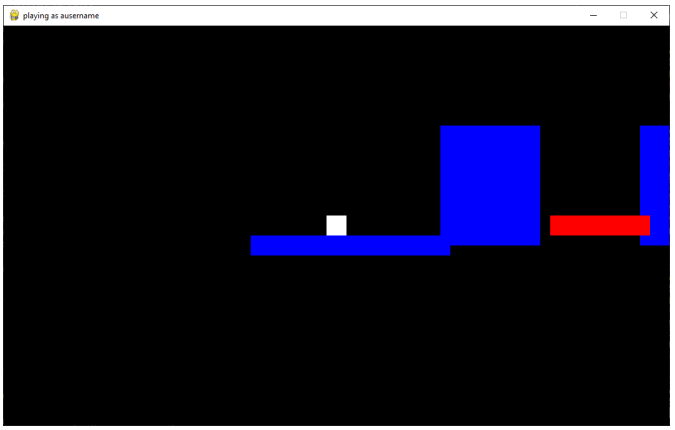


Fig E.22

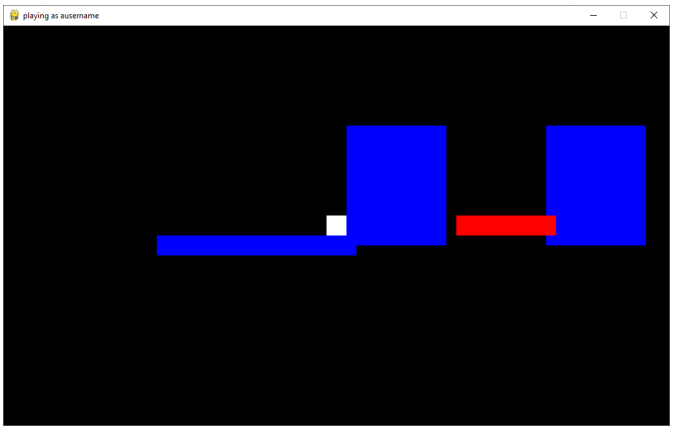


Fig E.23

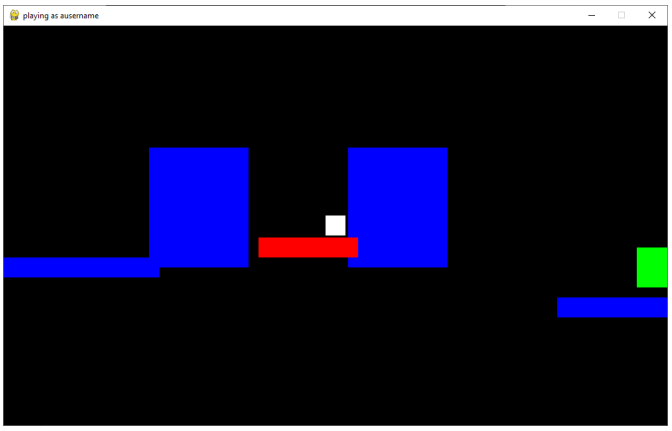


Fig E.24

Fig E.25

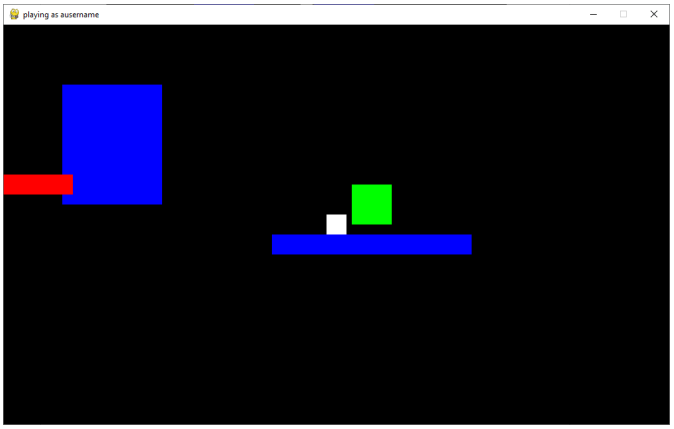


Fig E.26

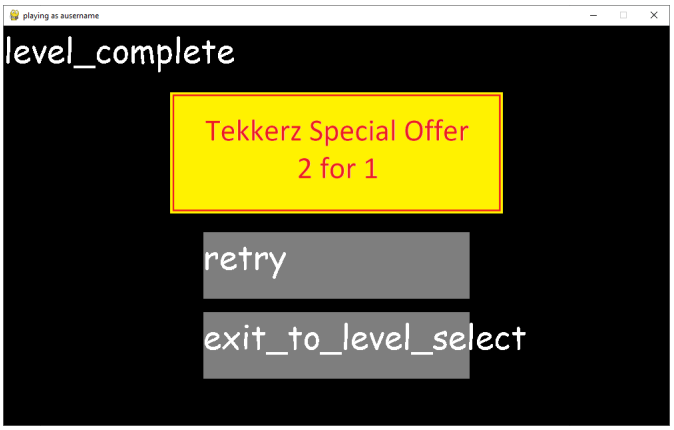


Fig E.27

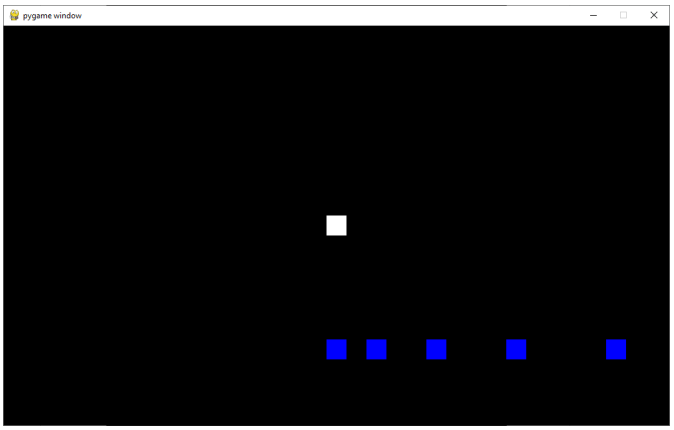


Fig E.28

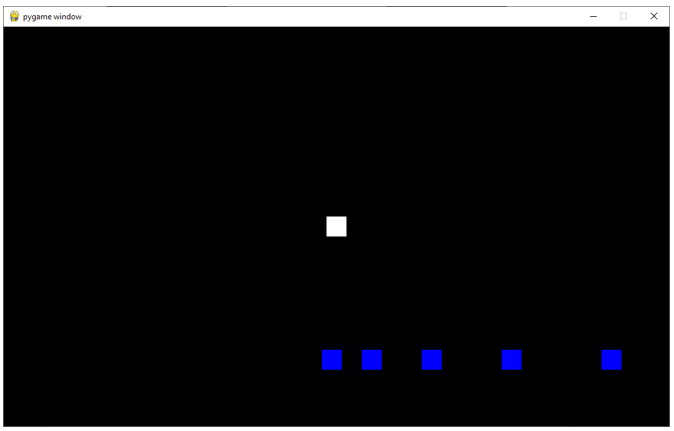


Fig E.29

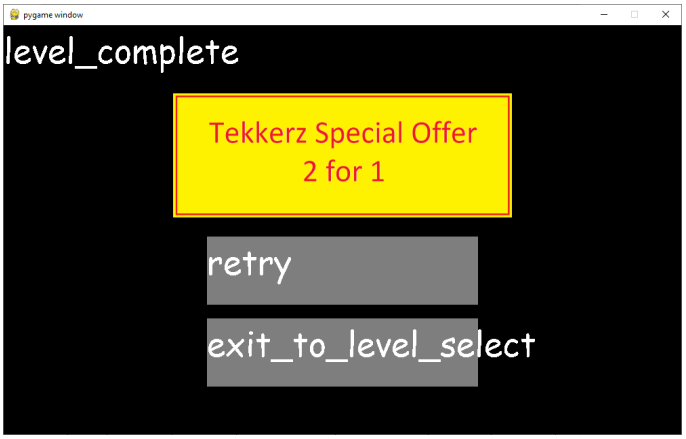


Fig E.30

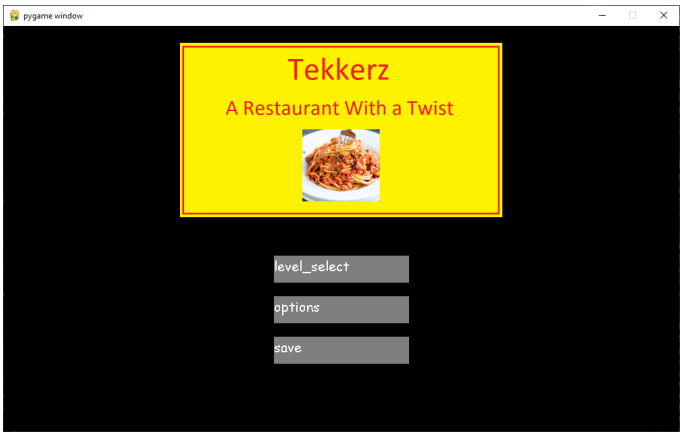


Fig E.31

Fig E.32

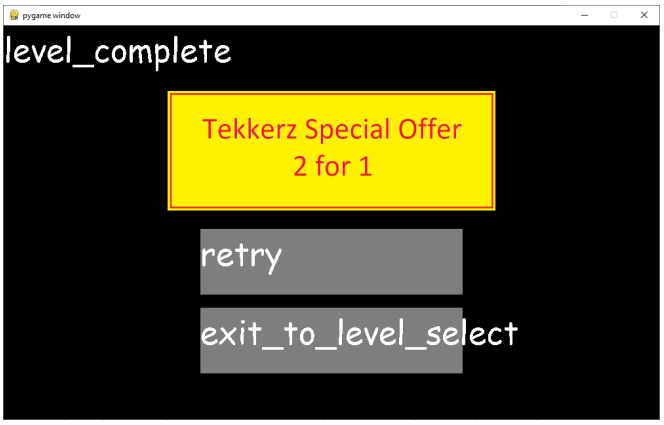

pygame window
level_complete
Tekkerz Special Offer
2 for 1
retry
exit_to_level_select

Fig E.33

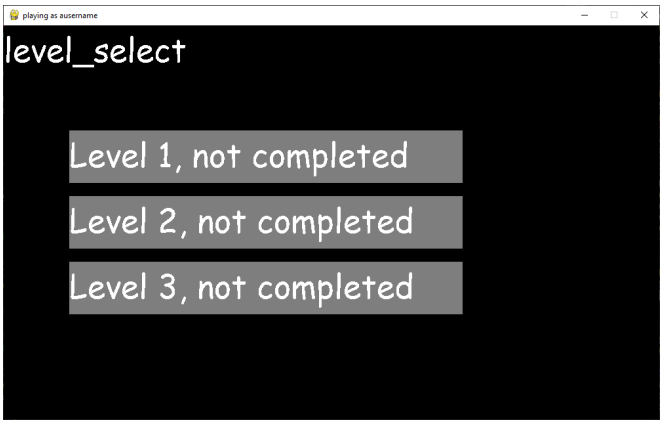


Fig E.34

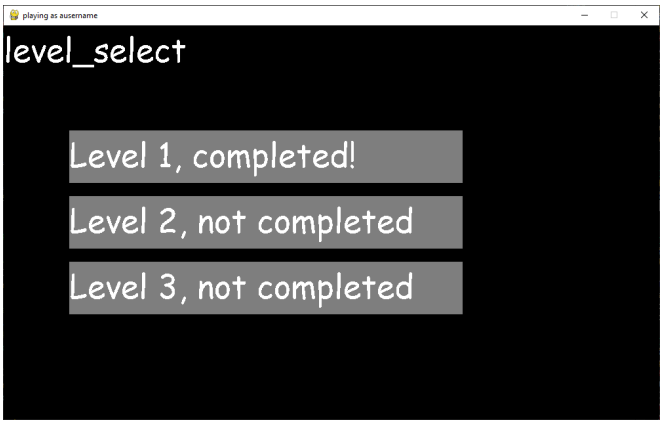


Fig E.35

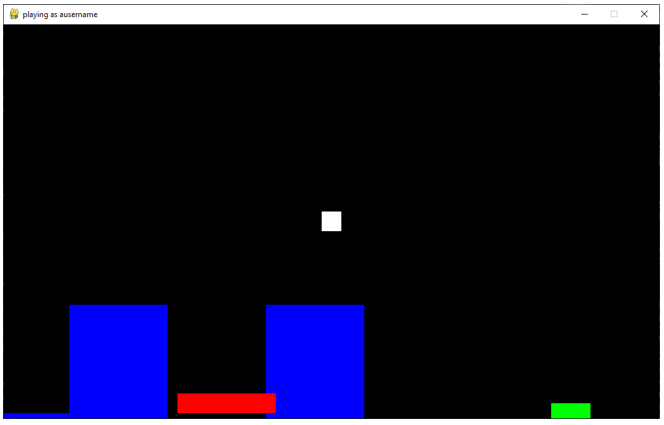


Fig E.36

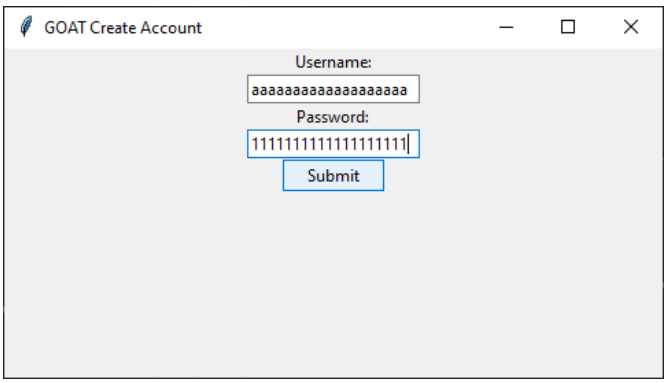


Fig E.37

```
��6������]�(�
users.user��User���)��}�(�username��
my_username��password��very_secure_password��
progression�}�ubh)��}�(h�  ausername�h�   apassword�h
}�ubh)��}�(h��aaaaaaaaaaaaaaaa
```

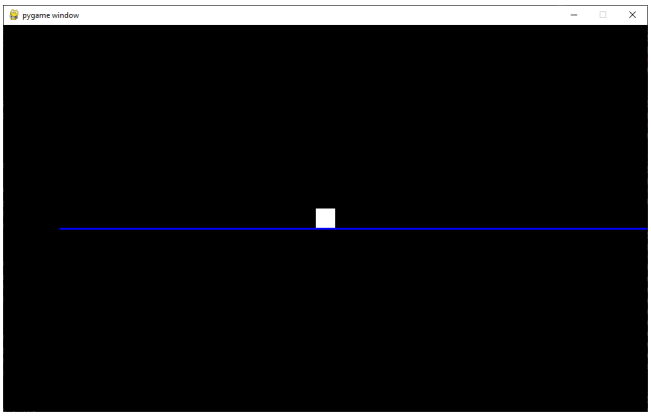


Fig E.40

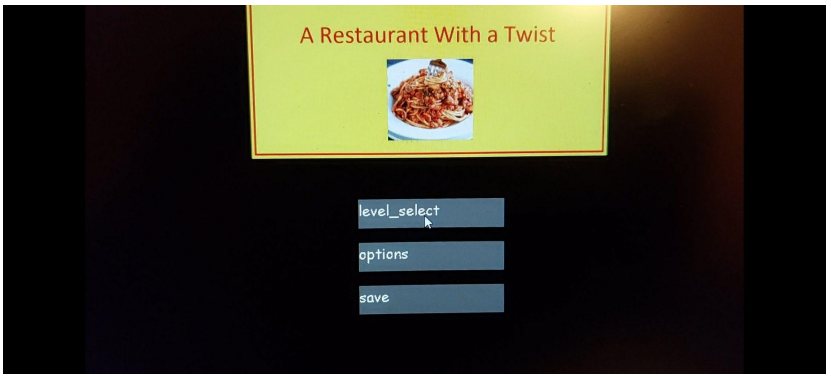


Fig E.41

Fig E.42

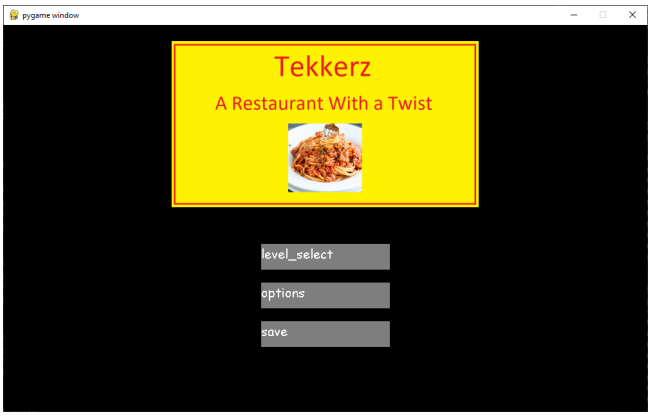
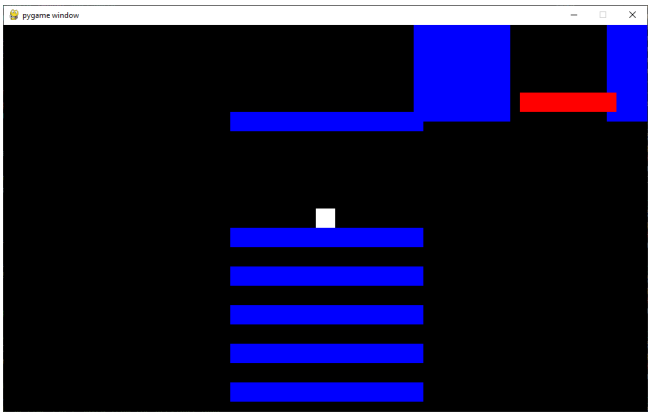


Fig E.43

```
{
    "x": 0,
    "y": 7,
    "hw": 5,
    "hh": 0.5
},
{
    "x": 0,
    "y": 9,
    "hw": 5,
    "hh": 0.5
},
{
    "x": 0,
    "y": 11,
    "hw": 5,
    "hh": 0.5
},
{
    "x": 0,
    "y": 13,
    "hw": 5,
    "hh": 0.5
},
{
    "x": 0,
    "y": 15,
    "hw": 5,
    "hh": 0.5
},

...
```

Fig E.44


pygame window

Bibliography

https://iqcode.com/code/python/pygame-boilerplate 18/03/2023 page 81
https://stacktuts.com/tkinter-boiler-plate 10/2022 pages 140, 149, 152
https://www.pygame.org/docs/ref/mixer.html 04/2023 page 223
https://www.pygame.org/docs/ 02/2023 throughout the project
http://jeffreythompson.org/collision-detection/rect-rect.php 04/2023 page 85