OCR GCE A-Level Computer Science Project H446-03

Name: Ben Fenocchi **Candidate number:** 7060 **Centre number:**15431
School: Wimbledon College **Title Of Project:** Developing “Cake Knight” The Game.

Contents page

Contents page 1

Analysis **4**

Description of the problem **4**

Why my solution is amenable to a computational approach **5**

Computational methods and approach **5**

Thinking abstractly, removing what isn’t necessary: 5

Thinking ahead: 6

Thinking logically: 6

Thinking Concurrently: 7

Thinking Procedurally & Decomposition: 7

Identifying suitable stakeholders **8**

Interviews: **8**

Summary of data retrieved from stakeholder interviews 9

Researching problems using existing solutions **10**

Summary of “Castle Crashers”: 10

Summary of “Ori and the blind forest”: 10

What I’ve learnt from these games: 11

Essential features of proposed solution **14**

Limitations of solution with justifications **16**

Hardware and software requirements **17**

Requirements 17

Justification 17

Software 17

Pygame libraries 17

Hardware 17

Measurable Success criteria **18**

Design **23**

Systematic breakdown of problem **23**

Defining structure of solution(using my main game objects): **24**

Player movement 24

Player attacking and taking damage 25

Loading levels - importing text files, extracting info and building levels 25

Enemies movement 25

Enemies attacking and taking damage. 25

In-game items - drawing and collision detection 26

In-game items, updating scores. Health bars and game info- drawing and updating these 26

Projectiles - calculating paths 26

Projectiles - destroying impacted tiles if explosive 27

Description of solutions using algorithms. Justifying their necessity in forming a complete solution. **27**

Player movement: 27

Player attacking and taking damage: 28

Loading levels - importing,extracting,building: 28

Enemies Movement: 29

Enemies attacking and taking damage: 30

In game items- drawing and collision detection: 30

Projectiles calculating paths: 31

Projectiles - destroying impacted blocks: 31

Ladders 31

Menus 32

Saving scores 32

Usability Features with justifications 33

Welcome / start menu: 33

Game screen (level hud): 34

Game over screen: 34

Instruction menu: 35

High Score screen: 35

Player experiences and difficulties: 36

Data Structures, variables, classes with justifications. **36**

Class diagrams: 36

Variable table with justifications 37

Files **43**

Input Validation **44**

Identifying and justifying test data **45**

Further data for post development with justifications **48**

Checking the final designs meet stakeholder requirements: **49**

Developing coded solution **50**

Documenting Milestones(with user feedback, testing & reflections for each milestone): 50

Outline of my iterative approach to coding the solution 50

Reasons for use of milestones: 50

Milestone Checklist 51

Success criteria checklist 51

Moving/jumping - basic level 52

Menus 57

Ladders/climbing 60

Enemies: 63

Health bars and taking damage: 69

Level objectives, coins and points system: 71

Attacking, crouching: 74

Importing level details from text files: 79

Ranged weapons 82

Highscores: 87

Sound effects: 90

Instruction screen: 91

Polishing features, sword animation, shield animation 94

Overview of errors encountered in developing my coded solution 96

Evaluation **97**

Testing to inform the evaluation: **97**

Annotated Usability testing: 97

Post development testing for function and robustness: 97

Evaluation: **101**

Success criteria cross referenced with test evidence. 101

Discussing failed tests from the “testing to inform evaluation section” 105

Addressing unmet criteria with further development: 105

Usability testing - stakeholder Joe’s review after testing 105

Evaluating usability features, justifying their success, Annotating usability features 106

The high score screen is very bland. The user feedback from my implementation section was that this could use some work. I would have liked to add some design elements like a trophy, or even some vines. 108

Ideas on increasing the usability of my game with further development 109

Considering Maintenance issues and Limitations: 109

Developing program to deal with Limitations and Maintenance 110

Does the project meet my requirements, key ideas I have noted/ overall review. 111

Bibliography/references: **112**

Final code listing: **112**

Analysis

Description of the problem

In recent years, most video games have become more and more complex, straying ever further from original genres like flash games or even arcade games. People have gone from playing games made by a single developer in months to ones made by teams spanning years. This increase in the man hours and technology invested into game creation has undeniably and inevitably led to an increase in the complexity of a game and the amount of time required to fully invest in a story line. I have asked many people what they missed most about games which they played when they were young, and the answer was almost unanimous. Many people yearn for the simplicity of a game with minimal story line, that they do not need to invest hours in for any result, that is fun to play, simple to use and master, whilst still retaining a significant level of difficulty to keep them entertained. I believe this description closely matches that of original platformers, therefore I think it would be a good idea to recreate one of these with modern resources and adaptations.

In 1981, Nintendo revealed what is widely believed as one of the first platformers, Donkey Kong. Blazing a trail for the millions of like games falling under the title of the popular game genre “platformer”. Since then there have been numerous advances in the field, due to the increased computing power of consoles and increasingly developed technology, and the genre, now sadly overshadowed by modern game engines and superior world design, from 2d side-scrollers to 3d open world maps, has reached a plateau, where the genre cannot advance any further from a technological standpoint.

The aim of most platformers is to navigate a series of platforms, hence the name platformer, either by jumping/falling onto them or by using strategically placed ladders, whilst working towards some goal, usually concurrent with the theme of saving an NPC or getting to the next level/difficulty, or surviving for as long as possible against increasing difficulty.

In this project I aim to modernise the original games like DK or Super Mario Bros, with better graphics and animations, now possible with modern computing power. I will still try and keep the classic feel, with pixelated animations and characters but will update the quality of them. I will expand on the functionality and usability of the original platformers by creating numerous levels, of varying difficulty, so the player can choose specifically the level of challenge they want rather than always starting off easy, and then getting gradually more difficult until the player dies, as is the case in many platformers. The main aim however is to make sure that the game is simple to play but still fun and entertaining, to try and help solve the problem of increasingly complex games, which require a more than desirable amount of devotion to the story line to achieve the games experience.

My next steps will be to research original and modern platformer hits like Donkey Kong, super Mario bros, ori and the blind forest, and note down their merits and, if they have any, individual failures to compile a list of what makes a good platformer. I will discuss this list with any

potential stakeholders or intended users, to decide which features will make the best edition for a platformer of my aim.

Why my solution is amenable to a computational approach

My game will have lots of functions coded into it. For this reason a computational approach is good as it will allow me to use:
Problem decomposition: I can break my problem down into much smaller subproblems(my functions) and work on these easier problems in turn and then put them back together at the end.
Divide and conquer: in a similar vein to problem decomposition, by dividing my larger task of creating the game into smaller sub tasks, these become easier to "conquer". This allows me to tackle large problems easily.
Abstraction: I will abstract away many unnecessary details to make my game easier to code in the timeframe. These include slanted surfaces (all platforms will be flat), and things like particle effects which I will not include due to my abstraction.

Computational methods and approach

My solution, being a computer game, naturally lends itself to being solved with a computational approach, and using computational methods. Therefore below I will outline each of these and how I will use them to effectively develop my solution, with justifications as to how they will be effective.

Thinking abstractly, removing what isn't necessary:

Thinking abstractly is the computational method of identifying necessary details and removing unnecessary ones. I have utilised this by trying to simplify my game as much as possible, in order to speed up development time and to let me focus on the basic functionality as best I can. It is better that I can provide a polished simple game to my stakeholders than a short complex one. As such I am including:

- -2D-platformer style game, simpler to implement. Easier to learn how to play.
- Simple menu, therefore easy to navigate.
- Few options to avoid overwhelming users, e.g. not asking to rebind keys, this won't be necessary as I will ensure they are very easy to use(My stakeholders are familiar with the usual use of WASD for movement in computer games).
- Simple health bar, no life system. I believe it is important to be able to monitor your health at a glance, and not have to try and remember how many lives or respawns you have left in a level.
- Simple user interface/HUD , this is achieved by using simple symbols to represent stats like health, ammo, money ect. These will be for example a coin to represent money, it

looks nicer than writing the word money on screen, and is also more visually appealing, and less crowding to the user.

This is justified as it allows me to divide my solution so I can create a checklist of simplified functions and features to implement, allowing me to set clear goals to work towards and ensure I don't fall behind in development by focusing on unnecessary features.

Thinking ahead:

It was necessary to think about my solution and what inputs and outputs it would need before I started to implement and code it, to avoid wasting time by making mistakes further down the line. I have decided that a mouse will be necessary for an aiming system, and a keyboard essential for inputs to control the player. Outputs will involve speakers and a monitor to display the level, menus and progress, with speakers helping to immerse the player.

It Is necessary to beforehand think about the wanted scale of the character in reference to the map, as this will affect the size of the sprites and the amount of freedom I will have in designing the maps. If I didn't plan this beforehand, I may have had to completely re-code parts of my solution in order to change the scales, wasting a lot of time.

Therefore I have come up with a scale in which a monitor of average resolution(1920x1080 which will be common among my target audience) will be able to fit 30 X 20 blocks which I will use to build my levels. The player will be as wide as one block and slightly taller, to avoid him being able to walk under them without utilising a crouching feature.

This scale will allow me to utilise enough blocks in the screen to build complex enough levels whilst ensuring nothing is too small to see.

I will have different block and background designs for levels and will also need a death screen/ resurrection animation for if the player dies. The death screen will allow the player the option of exiting to the main menu or trying the level again, or something similar to this, as is convention with most games. It will also be able to have the score displayed on it if I choose to.

Thinking logically:

I will split my game into separate loops. This is justified by allowing me to create scenarios for my different game states. For example I will have one while loop for when the Menu boolean is True- which will proceed to run the menu code, and one for the game itself, level select, highscores ect.

I need to figure out what loops I will have inside of my game loop, and what states the player themself will have. This is to ensure that in development I am clear about what states override others and will not make mistakes in this field, for example I need to decide that if the player is currently walking, but then starts to climb, the climbing will override the walking and result in the termination of horizontal movement, the walking animation will then be swapped to the climbing one ect. The order of priorities(highest to lowest) will be as followed:

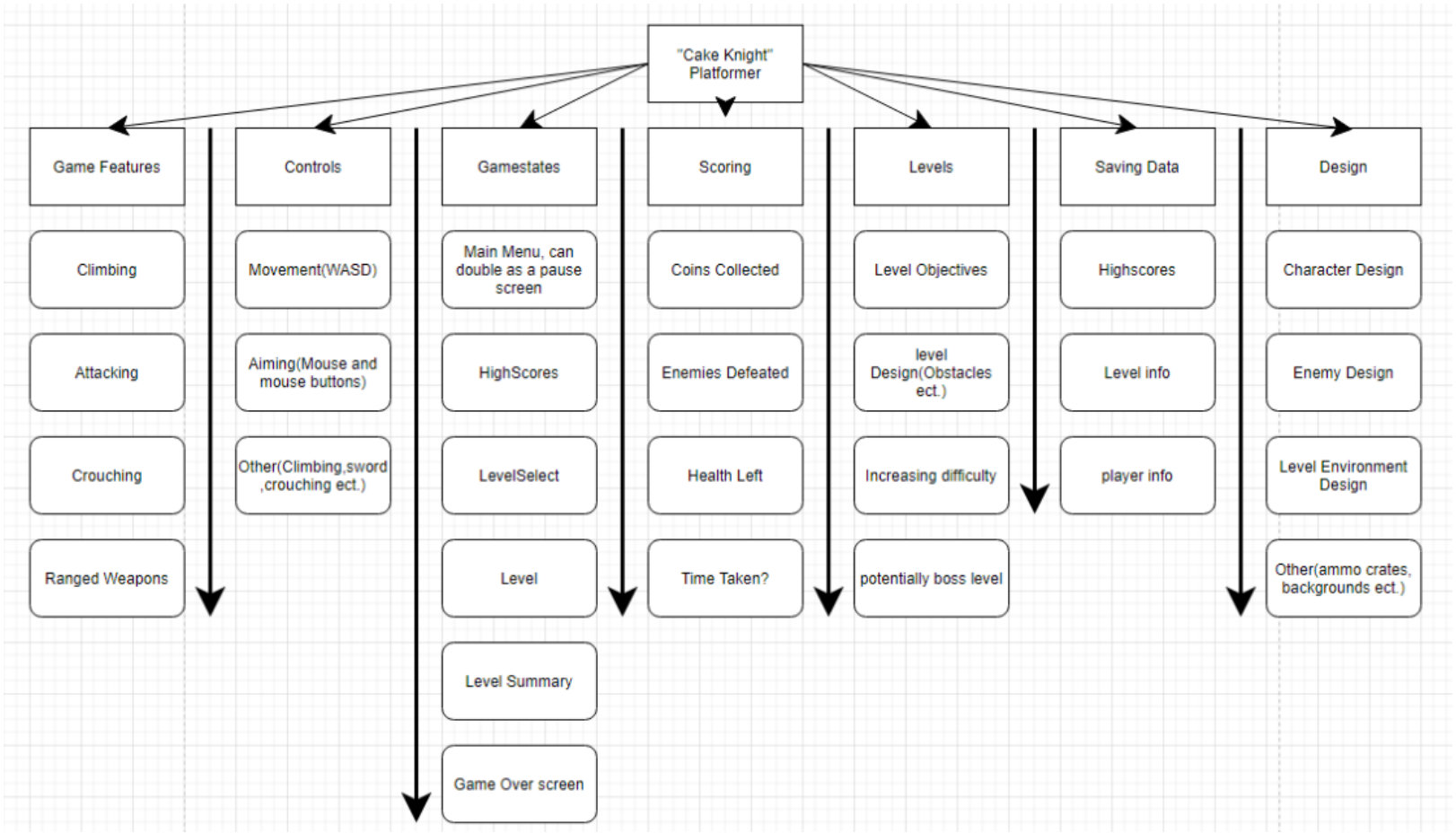
Jumping,Climbing,Crouching,Walking,Standing Still. This is justified as it will allow me to display only the necessary animation at all times. Jumping has to be in front of climbing to allow the player to get off the ladder ect.
By thinking logically I will be able to speed up development times by outlining any important conditions or loops.

Thinking Concurrently:

Thinking concurrently is the computational method of analysing the program and looking out for places where more than one part of your code can run at once.

I utilise this in my projectile movement code. This is because it has two movements, x and y and each is calculated separately, but in order to make everything look fluid, I change their x/y coordinates at the same time and then update the screen. This is great because it cuts out on unnecessary instructions,(making the game run slightly faster) and adds to a sense of fluidity which my stakeholders will enjoy.

Thinking Procedurally & Decomposition:



I have used procedural thinking in order to break down my solution into smaller segments. For example, my first iteration of code will give me basic inputs and player movement. Once this is

completed, I will then move onto a basic menu (that I can later improve upon), then ladders, then health bars, then enemies etc. This allows me to isolate smaller parts of my solution and work on them until they are complete without having to worry about how they will fit into other parts. My code will be neatly divided into each iteration and labelled, e.g. all movement code will be grouped together, creating the level grouped together, resetting values grouped etc., this will allow me to quickly find errors and make changes.

I used a top down modular design as seen above.

I have broken my solution down into 7 main parts, each with their own sub-parts. I have decided to do this as it will make my solution much easier to implement, by independently focusing on each part, and then linking them all together at the end like decomposition. I will do this with the use of functions and procedures in my code, for example my procedure to process and then implement the movement of the player.

Identifying suitable stakeholders

Selecting stakeholders

I will have two 17-year old's, one male and one female. These are people who are busy with examinations and revision but still need time to unwind.. The game will have PC's as the selected platform due to this increasing their playability. Users with already very little free time will often not be able to wait for a console to turn on and then start up the game, so it is necessary to be on a portable device like a laptop.

I have selected people to represent my target audience, to allow me access to the ideas and wishes of the target audience so I can better design my game for them. These people are 17-year-olds currently studying towards their A-levels named Joe and Charlotte. I will be able to regularly communicate with them in order to refine my current ideas for the game, and will set up interviews to help me do this.

Why the game is appropriate for their needs

They need the game in order to destress. With exams coming up soon, after spending most of the day revising, they will need a way to unwind and relax. I am going to create my game with the aim of providing this for them. It will be relaxing and also fun, and there will be no long storylines they need to sit through to play the game. They won't have enough time to engage in hour-long games with massive storylines, but will need something simple to help them relax. Therefore, this target audience is perfectly suitable for my solution of a simple yet fun to play platformer. Therefore my idea of an easy to play platformer will be the best thing for them.

Interviews:

In my interviews, I would like to determine what features they like to see in a platformer, and if they think a basic shop function is worth implementing. To do this I will ask these questions to members of my target audience:

1.Have you played many platformers?

2.What are your favourite features?

3.Would you like to see a shop function implemented?

4.what would you like to see in the shop?

5.Anything specific you would like to be implemented in the game?

6.What movement controls would you like?

Summary of data retrieved from stakeholder interviews

1.Most interviewees have played games like or similar to Super Mario Bros and therefore can answer many of the following questions. From this I know that the answers to the following questions will be similar to that of my target audience, so I can use the answers to make a better game for my audience.

2.Everyone said power ups e.g. Mario's fireballs. I can do this in the form of explosive arrows or bombs. Many people also said that they enjoy a multiplayer feature. From this I know that if I make explosive arrows(ones that can destroy tiles), my game will be more enjoyable.

3.Interviewees were mostly split on this. Some said it would be an interesting feature, they like the idea of using coins to purchase upgrades. Others said that they felt it wouldn't be necessary, and it is similar to the idea of power ups. This tells me I can not include it if I run out of time, and it won't upset my target audience if I have the powerup feature(explosive arrows).

4.The interviewees who liked the idea of a shop requested upon asking the following items/ideas to be present in it: A shield/extra life, weapon powerup (increased damage), explosive power ups for a ranged weapon. This clarifies the summary of point 3, a shop function won't be necessary for my users if I have a ranged weapon for them to use and enjoy.

5.Main ideas conveyed: levels, scoring system- possibly based on in-game collectable items, enemies, powerups. I now know that a scoring system will be vital to my game so I will make sure to implement it(using in-game collectable items as a way to get points) to ensure my users have fun.

6. Almost all interviewees request WASD controls instead of the arrow keys, they say that this is what they are used to using and therefore they will be most comfortable with. I will implement WASD to give my users their desired option.

From these points I now know what features to include going forward to provide the most enjoyable experience I can to my target audience. I will endeavour to include all of these to my solution despite the time constraints.

Researching problems using existing solutions

Current Solutions:

Summary of "Castle Crashers":

It is a side-on view multiplayer game where the player controls a character through pre-designed levels, defeating enemies and earning coins. These can be used to purchase in-game items to assist the player, with the aim of eventually saving all the princesses after beating all the levels. The game allows pausing and exiting to the menu however it only saves incrementally. The shop is available as an in-game level (with no enemies or objectives [you can leave it any time]), that you can access through the uniquely styled level select feature. Players can unlock pets with level progression or buy them in the shop, these pets have their own unique stats or abilities to help the player. There are 29 pets in the game.

Summary of "Ori and the blind forest":

Ori also has a side-on view. The aim of the game is to progress through a forest, occasionally picking up items/power-ups which may allow them to get to what were unreachable areas. The player is capable of jumping to different platforms and has to defeat enemies and puzzles. There is a somewhat unique checkpoint system in that the player can create "soul-links" at any point if they have enough resources, which become a checkpoint.

What I've learnt from these games:

Things I like from these games

What I like in these games	JUSTIFICATIONS - can I use these features?
Castle Crashers – Shop system	The idea of being able to purchase items to help you in-game is a great one. It allows you to introduce a new mechanic like money into the solution, making it more engaging and with more ways and styles to complete the game. Implementing this is possible, in the menu I could have a shop option which may take the character into a level, with a storefront background where they can stand in front of their desired items, possibly explosive arrows etc to purchase them. I would like to implement this into my game as I believe it would make it much more enjoyable, however there is more basic functionality to be implemented first, and this feature whilst it is desirable, will only be able to be added if there is time at the end of development. I am doing this to ensure the users will have all the necessary functions first so they will at the very least have a functioning game to play.

Castle Crashers – Aim of saving a princess	An Aim or simple backstory is relevant to a good game, this is because it helps build immersion, making for a more enjoyable experience. I could implement something like this to my solution, I want it to be light-hearted and fun, due to the age of my target audience, and so I can make the enemies look like cakes. This will make playing the game more enjoyable for my target audience as it will be fun and visually attracting.
Ori & the blind forest - Checkpoints	Checkpoints are good for longer levels, this is because it avoids the problem of the player dying, having to start over and becoming frustrated. However, while I like this idea I don't think it is necessary for my game as the levels are short and whilst in the more difficult ones there will be threats of dying, it should never be too much hassle to start over. I won't include checkpoints as I do not want to hassle my users with too many features, something that I am expressly trying to avoid in my solution to avoid overwhelming the player.
Ori & the blind forest - powerups	There are many power-ups such as the "spirit flame" that allows Ori to fire at enemies. Things like these would be good for my game as it allows for more play-styles and more fun ways to beat levels. However, some of them will be hard to implement or not serve much purpose due to my level design such as "Kuro's feather". The ability to glide will not serve

	my player as the map for each level is small and most jumps are possible and for those that aren't there will be a ladder system to help you to traverse the level. I will consider power ups but only ones that I believe will add to the user experience, again to avoid hassling the user with too many arbitrary features.

Things I don't like from these games.

What I don't like in these games	JUSTIFICATIONS - Can I improve on these features?
Castle Crashers – Pets	While this may be a good idea for bigger and longer games, this is not what my solution is about. I want a simpler game that people can play for a few minutes to have fun. These would not help with this as they introduce an aspect into the game that is too big and time consuming- hunting for or using different types of pets with different abilities. It is a complication that I feel is unnecessary and so won't be adding it to my solution in order to keep things simple and fun for my user.

Ori & the blind forest – Ability to *always* create checkpoints.	While this factor can increase the amount of potential levels and make for more fun and engaging levels, I feel the ability to always create a checkpoint (provided you have materials) may take away from the experience. This is because It may make sections designed to be hard easy, and remove some challenge. This could be implemented to my game but I feel it unnecessary because my levels are not long enough to require checkpoints, it would just add a complicated and also useless feature where it is important for my solution to be simple. This is what my stakeholders and interviewees wanted.

From this research I now have more ideas for what to include in my solution, I also know which of these features work and which don't. Combining this with my interview data I have created a list of essential features to include as seen below.

Essential features of proposed solution

-The character will have a run animation, and a standing still image. These will be flipped to form the images of the character going in the opposite direction, this allows for a consistent animation, and is less time consuming to implement than drawing and creating a separate run animation for left and right. It also means the player can see which direction the character is facing, this is important as it allows you to see where the player will attack if prompted.

-The character will be able to collect coins scattered around the level to increase their score, as well as by killing enemies, this adds another factor to the levels to be incorporated into the calculations of high scores, and will therefore lead to more interesting levels with different ways of completion with good scores. This is great as it allows my target audience to have many different playthroughs without it getting boring/ repetitive over time.

-The character will need to collect all the crystals in a level in order to progress to the next, this means that you can place the crystals in better guarded or harder to reach places, to vary the level of challenge to the player. This is justified as having a changing difficulty level is important to my stakeholders so they can always feel challenged.

-There will be powerups located in some of the harder levels, for example maybe a bow, or extra ammo for a bow that the player may have from the start, this will create more interesting

game mechanics and lead to a more enjoyable experience, as it introduces more strategic elements, in that it allows for ranged and therefore safer attacks, and its importance was made clear in interviews by my stakeholders.

-There will be 5 levels, progressively increasing in difficulty, with a level select option available. This is justified by the player being able to choose a difficulty best fitted for them, so they have a more suited experience.

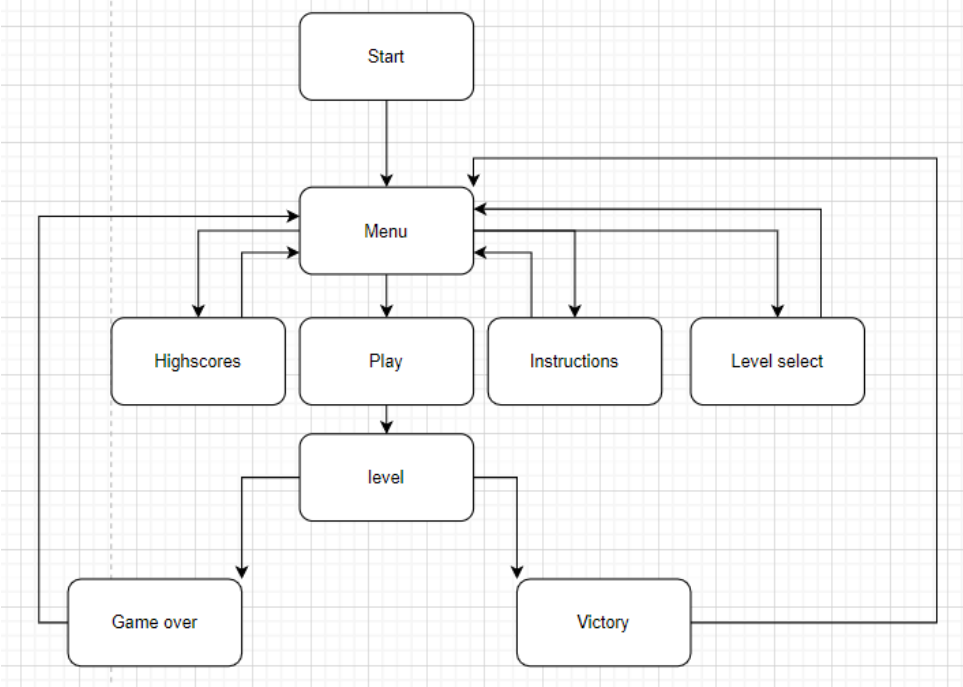
-Upon launching the game there will be an immersive menu screen, with an option to select another menu (play, level select, high scores) with arrow keys, as this is easier to implement than mouse clicks. This is justified as it will allow me to focus more of my limited time on other more challenging aspects of development.

-The standard enemy will be one that is confined to their floating platform, this is easier to implement and it also means that there will be no pathfinding algorithm necessary. The enemy will knock back allowing the player to strategically clear platforms of enemies, if they don't wish to risk more health in killing them. I may implement a ranged enemy, who will remain stationary and fire projectiles at the player. This is justified as I believe this will make levels much more fun for my users and provide a better experience.

-I will have different themes for the 5 levels, the first few will be set outdoors, then the last three will be inside, the platforms will be made out of rock instead of dirt, this will be reflective of the increased difficulty. This will also be more visually appealing to the target audience resulting in a better user experience.

-There will be a health bar as opposed to a lives system, this is because it will be very difficult to avoid taking damage, so lives will be more frustrating if the player dies and respawns each time, and to avoid this I will use a health bar to try and prevent players frustration.

-WASD will be used to control the players movement, as has been requested by my interviewees, as they feel that this is often the standard for PC games, and therefore the easiest to use due to the fact that my stakeholders will have had prior experience with it.



Limitations of solution with justifications

-I will not be able to implement high numbers of levels e.g. 50 This is justified due to the limited time and resources I have available to me: I will not be able to create large amounts of levels with enough individuality to avoid a repetitive feeling. I will have to make do with smaller numbers (around 5) as each has to be designed and created specifically to tailor for certain difficulty levels. However, in doing this I will have more time to focus on other aspects of my solution like in-game features such as ranged weapons - this is justified as my stakeholders have mentioned they will enjoy it.

-To address the problem posed by lack of hardware, the game will have simpler but still well-designed graphics, with less images to switch between in the corresponding animations. This is justified as I am limited by the hardware I and my clients possess, in order to have high frame rates, good graphics and even systems in place like lighting for my game, graphics cards and high-speed RAM will be needed(which my target audience won't all have.) This may provide a somewhat jittery feeling, but this is in keeping with classic platformers which is ideal for my solution, and something crucial to the desires of my target audience.

-Despite many people saying that they would enjoy a multiplayer feature, I do not believe that I will be able to implement this, my justification again being time constraints. This is because it would involve creating new keybinds for each new player, or possibly allowing controller input to

avoid too many people crowding one keyboard. However I do not have the skills to do this in the time available, so the game will likely be single player.

-Time and resources are a big limiting factor. Ideally my game would have a wide range of powerups, interesting features like shop functions and a wide range of levels, but I have to focus on the basics first and come to these things later time permitting, this is justified as it ensures that by the end of the project there is a suitable game for my target audience to play.

Hardware and software requirements

Requirements	**Justification**
Software	
Pygame libraries	My game makes use of modules from the pygame library, therefore this will need to be installed on the user's computer for the game to run, however, I can always install this code onto the final executable, in which case it will not be necessary.
Windows 8/10, MacOS 10.7 & above, Linux	The computer needs to have an operating system installed capable of running python, it should also be a relatively new OS in order to be optimised for a 64-bit operating system, to make use of the wider address buses, leading to more RAM and a faster running program in general, this will lead to a more fluid and enjoyable experience for my target audience.
Python 3.6 and above.	My program is written in python; therefore, python must be installed onto the computer in order for it to run, otherwise my target audience will not be able to play the game.
Hardware	

Clock Speed on processor- 2GHz+.	Need a minimum of around 2GHz, as for the graphics to run smoothly, the program will update the images in the animation, of all the sprites on the screen every second or so, this alongside of all the collision detection, redrawing the game window, and the map every frame, will require sufficient clock speed to avoid looking jittery. If it does look jittery it will create too much of a budget look and my target audience will become disinterested.
Hard drive: no more than 50MB.	All the images and files that the user needs to run the game will not exceed 50MB, so more than this is unnecessary.
Keyboard.	This is necessary to control the character, they are moved with inputs from the keyboard. Without a keyboard, my users will not be able to play the game.
Monitor.	Necessary to process the details of the game, to see what is happening currently. Again without this my users will not be able to play it.
Mouse.	Needed to select levels and use weapons such as a bow. A mouse will allow the users to be able to engage with the fun ranged weapons and have a more enjoyable experience.
Speakers/Headphones.	Background music and sound effects will be utilised for a more immersive experience, speakers or headphones will be needed to achieve this. This will create a more immersive experience for my users.

Measurable Success criteria

No	Success Criteria	Justification for criteria	Measurability of success

1	Must have WASD for movement, with the addition of spacebar for jump(as w will be for ladders.)	Charlotte has expressed her wish for these keys to control the movement of the player, due to their common use in pc games, This means she will be able to play my game easier and will have more fun, instead of getting confused over controls.	pass/fail
2	At least 5 different levels.	In order to keep the game interesting Joe has stated in his interview that it should have multiple levels, preferably 5 or more. In doing this I am also increasing the variety of the game for my stakeholders, so it doesn't get boring.	pass/fail Number of levels:
3	Levels increase in difficulty, with one very hard level at the end.	As the player progresses Joe has said that it will be important that the game remains challenging, and so I have decided incrementing difficulties will be necessary in the level to provide a continued level of challenge for the user.	Do they increase in difficulty pass/fail
4	At least two different types of enemy.	In my research most platformers have a wide variety of enemies, making for more interesting features and play styles. I want to incorporate this into my game so it can be as fun as possible for the user, and they can have a better experience.	pass/fail

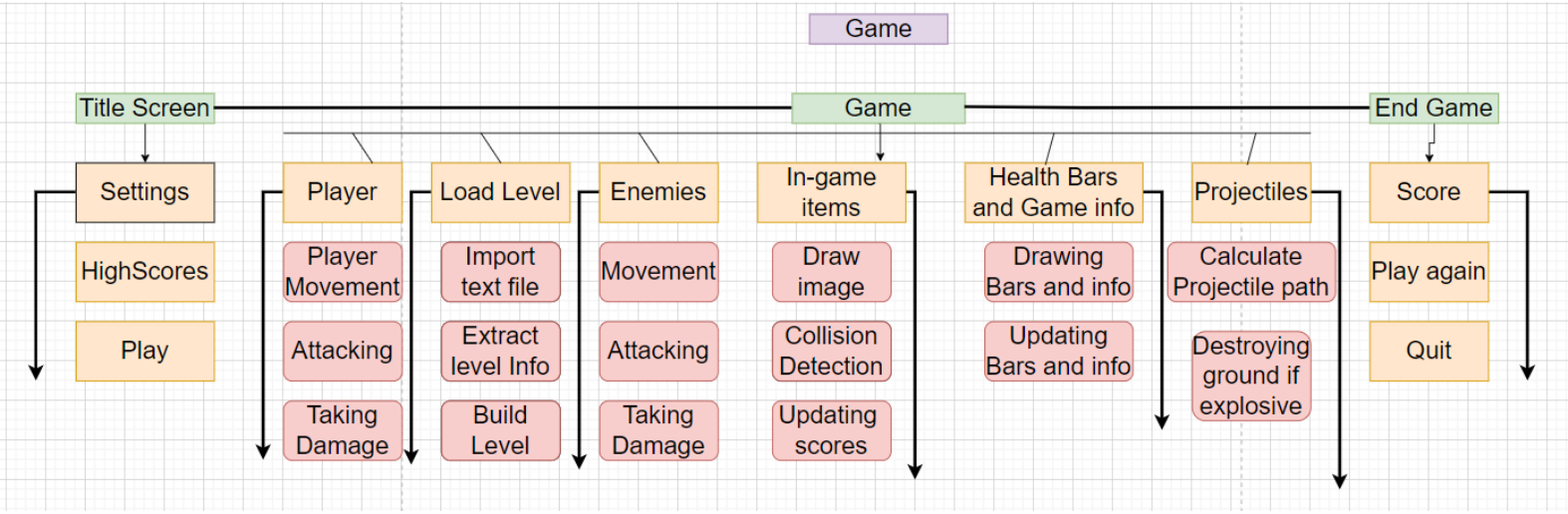
5	Power ups: at least one.	In their interview Joe and Charlotte stated that power ups should be integrated into the gameplay, in order to add new variables and play-styles to make it so there are many ways to beat a level, so they can be played multiple times without getting old. This is very important to my game as a problem with many platformers is they become boring after a few hours, as stated by many of my stakeholders, therefore I will make sure it is as interesting as possible.	pass/fail
6	A ranged attack that the player can use.	If the character has a ranged weapon, this will make for more interesting levels, and therefore contribute to solving my problem of keeping the player entertained.	pass/fail
7	A melee attack the player can use at close range. Time permitting, a cooldown on this weapon.	A basic attack before better weapons is discovered or bought between levels, to make sure that early levels are playable, and my users can have fun.	pass/fail
8	Give the player multiple lives or a health bar.	This ensures that they will not die immediately/ be able to play for a substantial amount of time and therefore not get frustrated.	pass/fail

9	Display lives/health bar.	This is very important for the player to know, as it will influence how they play the game. A health bar will be important as it stops the players getting frustrated with not knowing how much health they have left.	pass/fail
10	A menu with highscores, level select and play option.	Players should be able to see their high scores, or choose which level they want to play based on their abilities, this ensures they are always playing the best suited level for them, and so have the most fun possible.	pass/fail
11	Must have collectable coins in every level.	This can be used in calculating the high score, and maybe also time spent on each level. Adds another factor to the game as well as possibly being used to buy weapons/ upgrades. Joe said this was important in his interview, as it will make for more interesting levels, which means he can have more fun.	pass/fail
12	Sound effects and background music. At least 2 different music tracks to reflect difficulty.	Increases the level of immersion making for a more fun session.	Sfx: pass/fail Bg music: pass/fail
13	A way to score points by for example defeating enemies (cake soldiers), or collecting coins.	Will encourage players to go for the challenge of defeating enemies, in order to achieve a higher score. This element of challenge will make the game more captivating.	pass/fail

14	Have at least three crystals for the player to collect in order to pass the level.	This will mean that levels will not be too easily beaten, making for a more challenging experience, which will provide satisfaction to the users when they complete it.	pass/fail
15	Include ladders for vertical traversals too big for the jump feature.	Means level design has less constraints, making for better levels. Also adds gameplay possibilities like half climbing a ladder to avoid an enemy, which will provide more interesting playstyles.	pass/fail
16	Have at least one damaging block. E.g. lava the player can fall into or spikes they can walk on.	An extra detail for the player to be aware of, making the game more interesting.	Number of damaging blocks: pass/fail
17	Have many different platforms separated from each other and at different heights.	Forces player to make use of the jump ability or ladders, adds the extra threat of falling into lava, which will provide more of a challenge to the user.	pass/fail
18	Possibly include a menu to control sound settings, toggles for sound effects on or off etc.	This makes for a more customisable experience, leading to a happier player.	pass/fail
19	Game must be fun to play for the target audience	It must be enjoyable to play as otherwise it will not help my stakeholders.	Rating out of 10

Design

Systematic breakdown of problem



I have broken down my game in the following way to make it an easier problem to tackle, each box is one specific sub-problem, and when all are assembled/integrated they form my game.

Explaining and justifying my process

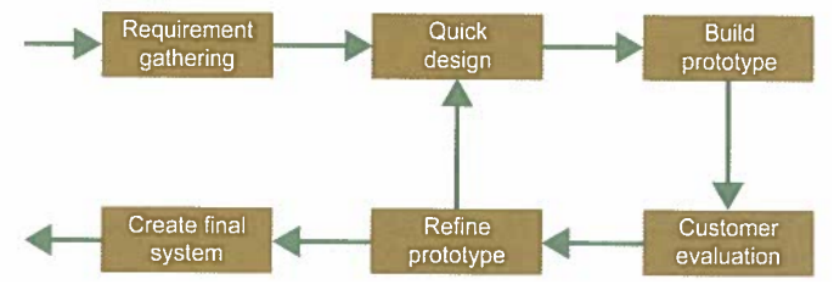
This way I can work on and test individual aspects, instead of worrying how all the code fits together when I am trying to debug. This is a form of problem decomposition and is a well known and used tactic in large development projects. This will speed up development as modules are now easier to code and therefore also quicker to debug due to their “isolation” from other modules. After completion of one module it will be integrated with the next module and so on, until I end up with a completed game.

Development methodologies:

I will use the waterfall methodology to outline my whole process. This is because it clearly breaks the process into a structured format for me to follow, and outlines requirements early on in the analysis stage, which I have done already, so I can have a clear process to follow and am therefore less likely to veer off track developing my solution. It also lets me set myself time frames for each stage of the project, so I can stay on track.

I will also utilise the technique of agile modelling, this is an iterative process in which incremental changes are made to each new prototype until there is a finished product. Below is

an **image** explaining the process **from the OCR AS and A Level Computer Science textbook ISBN:978-1-910523-05-6**



Throughout agile modelling, the user and stakeholders are integral to the process, this is because you need their feedback to refine the prototype for the next iteration. As such after each stage of my development process, I will have sections for user feedback, where I can use it to refine my solution.

This is great as it ensures that my solution is always being developed in line with my stakeholder needs, as the game is for them. Therefore it increases my chances of a successful project, and one that is better for my stakeholders and target audience.

Defining structure of solution(using my main game objects):

- ☐ Player: moving,attacking, taking damage
- ☐ Loading level: importing,extracting, building
- ☐ Enemies: movement, Attacking, Taking damage
- ☐ In game items: drawing, collision detection, updating scores
- ☐ Health bars and game info: drawing and displaying, updating
- ☐ Projectiles: calculating path, destroying ground/ dealing damage

Player movement

If the variable moving_left is set to “True” then the player should move to the left as this means the key “a” is being pressed by the player. Therefore the value 5 (speed of the player if you will) is added to the array player_movement which is responsible for holding details of the players speed in each direction. If the player is not crouching or jumping, the walk animation will be updated by displaying the next image in the cycle.
The same is true if moving_right is set to “True” but the player will move in the opposite direction.

Justification:
There is a walk animation because it helps the game to look more realistic and therefore be more immersive. It also adds to the aesthetics of the game, and makes it look more like the platformers that I am trying to emulate in my solution, to also bring a sense of nostalgia to my stakeholders

Player attacking and taking damage

The player will be represented by a “Rect” object courtesy of pygames inbuilt rect class.
When the player is attacking, the rect object representing the sword will check for collisions against a list of all the enemy objects(they are in a list so this can be done in an iterative process and be completed much quicker), and if there is a collision then will deal damage to the resultant enemy. Taking damage works in the same way for ease of coding, where the sword is essentially the enemy.
Justification:
I will continue to use rect objects for the following reason: It has a library of collision monitoring methods, and will save me lots of time in checking the coordinates of the player and comparing them to every other ingame object. It will also make the game run much faster as the methods have been optimised for such purposes.
The player needs to be able to attack and take damage in order to interact with the game, and progress through the levels collecting points. This lets them complete levels, and conquer enemies leading to a sense of satisfaction.

Loading levels - importing text files, extracting info and building levels

A level represented in a text file will be imported into the program, then taken apart character by character, line by line, the in-game items represented by the character will then be placed into a 2D array, which will then be composed of all the information required to build the level. This is now also essentially a map of the level, which can be used to check the players position relative to objects, and even updated if needs be(e.g. a coin is collected so will be removed from the map).
Justification:
I have decided to represent the level in a text file as it is much easier to create and design a level rather than inputting levels directly into 2D arrays. It also makes my code much easier to read and trace through, as there aren’t large portions disrupted by massive arrays.

Enemies movement

When moving my enemies I need to make them be able to change direction if they are about to fall off an edge or hit a wall. As such I will need a variable to hold the direction of movement. There are also many other variables they will need, such as potentially class,speed, health ect.
Justification:
Due to the fact that there are also multiple enemies per level all with their own variables (or in this case attributes), the problem lends itself naturally to OOP. Therefore I will have an enemy

class with a method specifically for movement that will reverse the direction of the enemy_ facing attribute if they are going near a cliff or wall.

Enemies attacking and taking damage.

If an enemy comes into contact with a player they will deal damage equivalent to the damage attribute of that instance of the enemy class. As such I would need a way to verify contact with the player and so each enemy will have their own rect class which will inherit the enemies specific x,y, width and height attributes. Then collision detection will be easy as it can use the same system as the player, pygames in built colliderect() method.
Justification:
I will be using this method as it makes the process easily iterable, saving much time in saving me from manually checking for collisions based on the coordinates of each enemy.

In-game items - drawing and collision detection

Each item will need to be redrawn along with the background and every sprite each frame to stop a phasing effect(to get rid of past drawn images like the player at previous occasions).
Justification:
As such it makes sense to represent each sprite/ in-game item with a rect or dimensions equal to or similar to the image dimensions, to handle collisions, and then draw the image on the coordinates of the rect object each new frame, as this makes the collision detection much easier, saving me time and allowing me to focus on other areas of development.

In-game items, updating scores. Health bars and game info- drawing and updating these

Each time you collect an item- whether it be ammo, coins, crystals or take damage, the corresponding score/counter that will be at the top of the screen will need to be updated.
To do this you will need to have a variable storing the value so that this can be projected onto the screen. However the player will not be able to interact with/ bump into these counters and therefore there is no need to represent these with a rect object to handle collisions, as there will be no collisions.
Justification:
I am updating these scores so that the player will have all of the necessary information to play the game, and make it more enjoyable. (It would be very frustrating not being able to see how much ammo you have left)

Projectiles - calculating paths

When a player fires a projectile, its trajectory will need to be calculated. This will have to depend on the position of the cursor and the player, as this is the only information that should influence the flight path(apart from possibly the class of projectile - however I think there will only be one of these to avoid too many unnecessary features.)
Therefore the function to calculate the flight path will need to take x,y positions for the player and cursor as parameters, and from this return the starting horizontal and vertical velocities for the projectile, which can then be used to move the projectile(again represented by a rect object

with an image drawn on top. (This is as usual for ease of collision detection and maintainability of code which will be particularly important as this will be a very large program compared to others I have made in the past)
Justification:
I will have to calculate the paths of the projectiles, using the knowledge of my maths and physics courses, to make their paths look realistic. This is because it will look more appealing and life-like to the user, leading to a more enjoyable experience.

Projectiles - destroying impacted tiles if explosive
This will happen if the projectile collides with a tile i.e a portion of the floor/platform.
To do this I will need a way of determining which tile the explosive hits if any, and then I would have to delete it from the level info array mentioned previously. To do this I will check for collisions with the projectile against an array of all the rect objects representing tiles, and if there is one then delete the corresponding tile from the level array.
Justification:
I have chosen to do this as it creates more interesting functionality for my game. For example, as enemies will not walk off a cliff, you can restrict their movements by destroying the tiles in front of them. This is particularly useful as it means I have more freedom in level design, I can create levels where it is necessary to restrict enemies for example, leading to the utilisation of problem solving which will be fun.

Description of solutions using algorithms. Justifying their necessity in forming a complete solution.

- ☐ Player: moving,attacking, taking damage
- ☐ Loading level: importing,extracting, building
- ☐ Enemies: movement, Attacking, Taking damage
- ☐ In game items: drawing, collision detection, updating scores
- ☐ Health bars and game info: drawing and displaying, updating
- ☐ Projectiles: calculating path, destroying ground/ dealing damage
- ☐ ladders
- ☐ Menu screen
- ☐ Saving score

Psuedocode algorithms	Justification

Player movement:
 	
IF movingRight THEN
  Player_movement[x] = 5
  Player_Image = animations[currentNumber]
  currentNumber+=1
	-Needs a line to check which way the player is moving.
	
	-Player_movement[] will hold all of the movements in each direction, this is because it allows me to easily set each movement speed to what I want.

lastFacing = “Right”
ENDIF
 	
The pseudo code will be the same for moving left, except with a negative value for player_movement[x]
 	
playerYspeed +=0.4
IF playerYspeed >10 THEN
playerYspeed =10
ENDIF
	-Last facing variable to let me know where the player was last moving, so I know which direction to make them face when they stop walking or try to use weapons
	
	-playerYspeed is added to each second to model the effects of gravity. It is capped at a certain value. This is what will allow the player to move down between platforms.
	These are needed to allow the user to properly move and interact with the game

Player attacking and taking damage:
Player_rect=[startx,starty,pWidth,pHeight]
 	
For all enemies in enemy array:
If enemy collides with player THEN
Playerhealth - 10
ENDIF
IF isAttacking THEN
sword[0] = player_rect[0] + swordLength
ENDIF
For all enemies in enemy array:
If sword.colliderect(enemy) THEN
enemy.health -=10
ENDIF
	-player_rect will be the rect object representing the player
	-for each enemy in an array holding all of the enemies. By storing enemy objects in an array, I can very easily check all of them for collisions.
	-if they collide, take 10 off the players health, the value of 10 is subject to change
	-By storing the sword as a rect object also, I can set its x and y co-ords to start on the player, so it appears as if they are holding the sword.
	-if sword touches enemy, take 10 off of its health, 10 is subject to change.
	
	These are all needed to allow the player to attack and ultimately defeat enemies.

Loading levels - importing,extracting,building:
DEF FUNCTION importLevel(levelNumber):
file = open(levelNumber,"r")
x = 0
for line in file:
content = line.split(",")
for i in range(len(content)-1):
levelArray[x][i] = content[i]
x +=1
return(levelArray)
END FUNCTION
	-Importing the level and extracting the information will take the form of a function(so I can repeat the process with as many different levels as many times as I like). I will import the text file and then copy each character in the file separated by a comma into a new array called levelArray(this name may change). At the end of the function this array is returned so it can be stored as a separate array with a proper name like level1Array

```
y=0
for row in levelArray:
  x = 0
    for tile in row:
      IF tile == "o" THEN
        win.blit(cloudImg,(x*TileWidth, y * TileHeight))
      ENDIF
      IF tile == "p" THEN
        win.blit(cloudImg2,(x * TileWidth, y * TileHeight))
      ENDIF
      IF tile == "w" THEN
        win.blit(grassImg, (x * TileWidth, y * TileHeight))
      ENDIF
      IF tile == "s" THEN
        win.blit(dirtImg, (x * TileWidth, y * TileHeight))
      ENDIF
      IF tile == "i" THEN
        win.blit(stoneImg, (x * TileWidth, y * TileHeight))
      ENDIF
     IF tile == "k" THEN
        win.blit(cobbleImg, (x * TileWidth, y * TileHeight))
      ENDIF
     tile_rects.append(pygame.Rect(x * TileWidth, y*TileHeight,  TileWidth,
TileHeight))
    x+=1
y +=1
```

This iterative loop will go through the whole level array line by line, and for each item(letter representing a tile) in the array, the corresponding image will be drawn on the screen in the correct place- thus creating the level. There may also be new tiles created in future, this is a current estimation of available tiles (ladders not present to save time, however they will also be included).
However- this is just the image, the rect objects representing the image also need to be stored so they can be used to check for interactions/collisions
To do this I will add them to an array of rect objects. I will be using an array so I can easily run through the whole list with a simple loop.

The above is all necessary to load level details and draw them whilst saving information on the where-abouts of the level components that will be interactive(e.g. floor), for later use.
P.S clouds will also be interactive, so I have put them in this loop.

Enemies Movement:

```
DEF PUBLIC PROCEDURE move(self):
        Collisions = collisionCheck()
        IF collisionsEnemy["bottom"] THEN
            self.movementy = 0
            self.counter = 0
        ENDIF
        IF  not (collisions["bottom"]) THEN
            self.counter +=1
        ENDIF
            IF self.counter ==3 THEN
                self.rect[1]-= 1
                self.rect[0]-= self.movementx
                self.movementx = self.movementx*-1
            ENDIF
            IF self.movementx>0 THEN
                self.facing = 1
            ENDIF
            ELSE
```

This movement code will be a method within the enemy movement class so I can re-use the code in any subclasses I might create.
There will be a dictionary called collisions that will store info on all the possible collisions. If for example the enemy collides with the top of a block, then we know it is walking on the surface and we will make sure it stays at the same y-level. I will store this in a dictionary as it is easy to use being a single data structure that can store info on multiple things.
The if not Collisions[“bottom”] clause is there to make sure the enemy doesn't walk off of a cliff. If it does the y value will be reset and the direction of movement reversed.

self.facing = -1
IF(collisionsEnemy["right"]) or (collisionsEnemy["left"]) THEN
self.movementx=self.movementx*-1
self.facing = self.facing * -1
ENDIF
	This is to make sure the enemies are confined to their platforms to make the game more fun. It also makes sure the enemies won't fall into lava. This is good because if they do there is no point in having the enemies, and therefore it removes a fun element for my users.

Enemies attacking and taking damage:
for all enemies in enemy array:
IF sword.colliderect(enemy) and playerDealDamageCounter == 0 THEN
enemy.health -= 50
enemy.rect[0]+=10*lastFacing
ENDIF
for block in damageBlocks:
IF block.colliderect(enemy) THEN
enemy.health -=10
ENDIF
IF enemy.health <=0 THEN
Remove enemy from enemy array
enemiesKilled += 1
ENDIF
playerDealDamageCounter +=1
IF playerDealDamageCounter >= 10 THEN
playerDealDamageCounter = 0
ENDIF
 	
Enemy attacking will have the same functionality as the player taking damage code above.
	Enemies health will be an attribute of the enemy class to avoid having to manage too many variables. If the enemy collides with the player's sword, any damaging block e.g. lava the player will lose health. If the enemy dies they are removed from the array holding all enemies and their information(deleting the enemy) and a kill counter is incremented.
	This is so the player can see their scores/ achievements and therefore make the game more enjoyable.
	The damage counter is to ensure damage isn't dealt too fast as instant kills wouldn't be fun. This is because the computer runs through the code many times per second, so without the counter the enemies would appear to die instantly. I have learnt the importance of this in past coding projects and so I know I must include it when I begin to develop my game.
	
	All above works to ensure that the enemies can be damaged, and damage the player, making for a more challenging game for my users.

In game items- drawing and collision detection:
for crystal in crystals:
IF player_rect.colliderect(crystal) THEN
crystalsGathered +=1
Remove crystal from crystal array
Remove crystal from the level array
ENDIF
	This is an example for one in game collectible object: crystal. However the code will be very similar for all objects for ease of programming and code maintainability
	If the player touches the crystal, their score will be implemented, the corresponding crystal in the level array will then be located based on its x and y positions, and deleted to make it disappear from the screen.

	This also ensures it cannot be collected again, otherwise levels will be able to be easily completed and this will not be fun due to a lack of challenge to getting points for my users.
Projectiles calculating paths: aimDeltax=mousex-(player_rect[0]+player_rect[2]//2) aimDeltay=mousey-(player_rect[1]+player_rect[3]//2) Hypotenuse =square root(aimDeltax**2 + aimDeltay**2) Velx = aimDeltax/Hypotenuse * speed multiplier Vely = aimDeltay/Hypotenuse * speed multiplier	Hypotenuse finds the total distance between the centre of the player(where projectiles will appear from) to the mouse. The x velocity is the x component of this distance compared to the full distance multiplied by a scaling factor to make sure the speeds aren't too high or small. The same for the y component. This is to make sure it is easy to aim the projectiles, their path should relate to the position of the mouse when fired. This will make the game more enjoyable Each cycle, the y vel should be increased(because of gravity) to make it resemble an arc of a projectile in real life. This is to make the game more immersive as it will make the projectiles seem more realistic to my user.

Projectiles - destroying impacted blocks:
for each destroyable block:
IF projectile.colliderect(block) THEN
Remove block from level array
ENDIF
	To do this I will need a way of determining the block that the projectile hits, and then I can delete it in the same way I delete in-game collectable items as seen in the in-game items section.
	
	However I will need a destroyable block array, as not all blocks should be able to be destroyed such as lava or ladders as these may be essential to the completion of the level.
	
	This is to ensure the game remains playable and is therefore still fun.

Ladders	If the player presses w then I will set the climbing boolean to true. Then later on in a loop with a

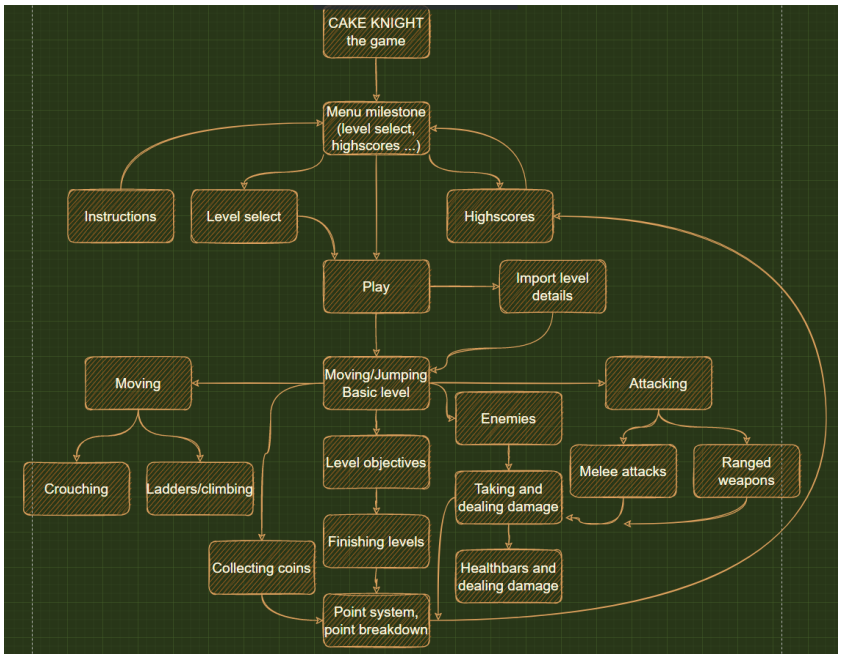
IF keyW THEN
  Climbing = True
ENDIF
IF Climbing THEN
  movingRight = False
  movingLeft = False
  FOR every ladder
  IF player.colliderect(ladder) THEN
  Player_movement[y] =-5
  ELSE
  Climbing = False
  ENDIF
ENDIF
	condition set to the value of this boolean, I will set left and right movement to false, so the player stays on the ladder, and then if the player is touching a ladder(this makes sure they cant climb when they shouldn't be able to), make them go up by decreasing the y coordinate, (top of the screen is zero in pygame)
	
	Ladders are essential to my game as it allows the player to traverse to any otherwise unattainable platforms, thus making the game more interesting as it expands the play area.

Menus
Choices = [play,levelselect,highscore,help]
WHILE menu:
  currentChoice = choices[0]
  Displayscreen(MenuSplashScreen)
  IF keydown AND currentChoice>0THEN
  currentChoice -=1
  ENDIF
  IF keyup AND currentChoice<len(Choices)THEN
  currentChoice +=1
  ENDIF
  IF keyEnter THEN
  Chosen = Choices[currentChoice]
  Menu = False
  ENDIF
ENDWHILE
Chosen = True
 	
WHILE Chosen:
  Displayscreen(Choices[chosen])
  IF keyP THEN
  Chosen = False
  Menu = True
  ENDIF
ENDWHILE
	Inside the menu while loop, the user will be able to alter the currently selected choice with the up and down keys, when they are done they will be able to press enter, this will finalise the choice. When enter is pressed the menu loop is exited and the current choice is finalised, so you can enter the next corresponding menu loop.
	
	This is necessary as it will allow the player to progress to the next part of the game that they want to/need to.
	
	Then in the next menu there will be an option to exit with the key press p, this will return to the main menu, allowing them to navigate the menus easily.

Saving scores
IF playerHealth<=0 THEN
  file = open(highscoresFile)
  file.write(playerScore)
  file.close()
ENDIF
	This code will save the players score to a text file if they die(health goes below zero)
	This allows me to then create a highscores feature, which will provide a competitive element to my game so it is more fun to play for my

	stakeholders.

These algorithms when put together should form a complete solution to my problem, as they are just the different aspects of my game broken down into modules.
Therefore once reassembled it will be the full game.
Below I have included a display showing how these algorithms will link together.



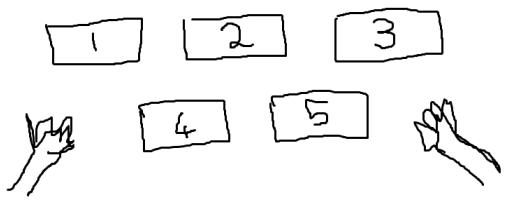
Usability Features with justifications

My game will have three main interfaces as is typical with most computer games. This is to ensure that the player will feel comfortable and will be able to easily navigate my game to create an enjoyable experience.
These interfaces will be a welcome/ start menu, the actual game screen, a game-over screen along with an instruction and high score screen.
I have drawn some preliminary pixel art designs for the more main features, and early MS paint designs for the aspects that are more likely to change, to avoid wasting time.

Welcome / start menu:

This is a current idea for my welcome screen - I may adapt it to make it more visually engaging.
It has a play option to get to the game, a level select for people to choose their starting difficulty. This is to

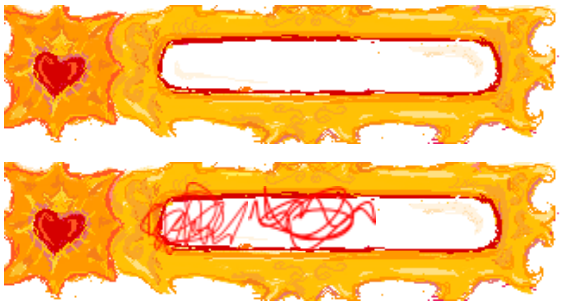
ensure that everyone can play the difficulty that best suits them for the best experience.



This is an initial level select screen I have created that is subject to change.
It would be a good idea to fill the white squares with level previews. This is to make sure that you can gauge what each level will be like from within the select screen, thus making sure players can quickly navigate to what they want.

There will also be a high score screen to give the player something to work towards. This is great because it will keep them engaged.
The high-score will be a table of the best scores, along with maybe names or the dates they were achieved. I will also make sure it is visually appealing with a design scheme similar to the above images, this will make it aesthetic for the user.

Game screen (level hud):



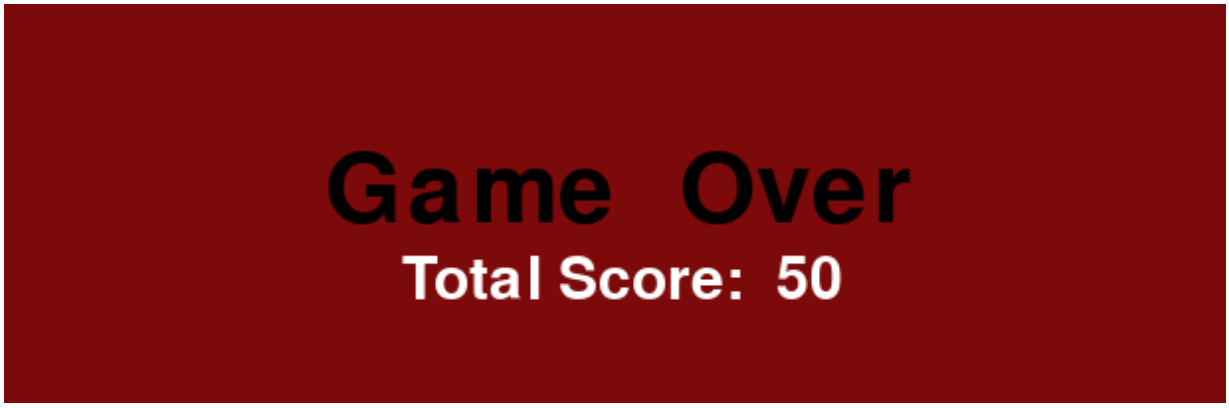
The health bar will be visible at the top of the screen, this is so players won't get frustrated by dying suddenly, but will have a warning. I will also make the colour of the health bar go redder as the player's health decreases, this will make the game more immersive. However in case members of the target market may be colour blind for example I also plan to make the health in the vial decrease as illustrated in this design.

Various symbols will be displayed, eventually along with the counters that they represent. The ammo will have a x5 next to it for example, to show the player their ammo. ***These are preliminary designs and may change later in development.*** The crystal counter is displayed to show you your progress through the level, enemy kill count and coins collected will be displayed to give you some indication of your score- this will encourage you to beat highscores if you know you are close. The ammo counter (projectile counter but I think bullets best

represents this) will be shown so the player wont get surprised when they can no longer fire projectiles.

Game over screen:

This screen will be very simple. It will inform the user that they have died and therefore the game is over, it will also display their score. This is so they know if they have beaten the high score or not. The screen will also drop down from the top. The current idea I have for this looks like this.

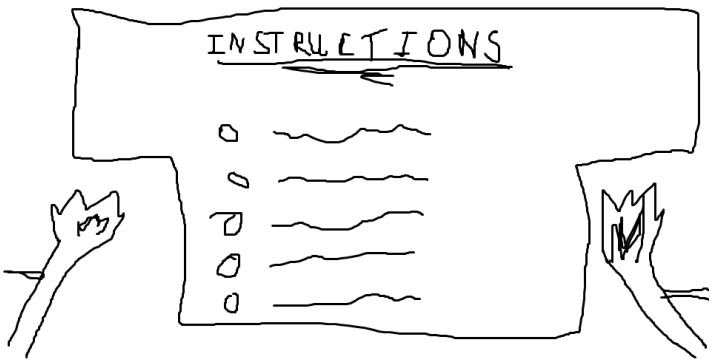


There is no option to restart as I will make the program return the player to the main menu where they can re-attempt from there. This is because they may want to choose a different level with an easier difficulty and they can do this from the menu(by going into the level select setting).

Instruction menu:

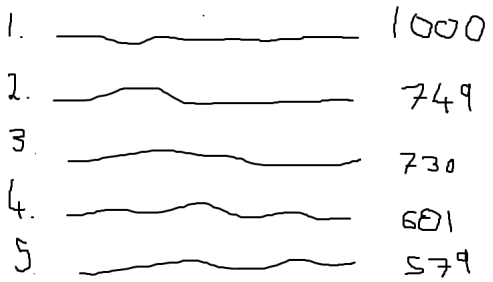
I will have an instruction menu so new users can see how to play the game and understand the objectives and controls. It should outline all the keybinds and the aims of the game. Below is a preliminary design for my instruction screen. I also have this checklist that needs to be completed.

- ☐ Controls for interacting with the game
- ☐ How to complete levels/ aim of the game
- ☐ Aesthetic simple design to appeal to my target audience



High Score screen:

The high score screen will be very simple. It should display the top five scores, in descending order. Given time I am thinking about including some effects around the first place, or a trophy because this may look a little bland. Maybe some flaming torches like above in the instruction screen design.



Player experiences and difficulties:
Increasing level difficulty will be achieved by varying the number of enemies, enemy statistics(health/damage/speed), types of enemies, amount of ammo available, locations of crystals to be collected and general map terrain. I will use this many factors to alter difficulty as it will ensure that subsequent levels can be harder, without feeling too repetitive or derivative of the previous levels. This is all with the aim of creating a better player experience.

To avoid bad game experiences such as starting the level and immediately dying or losing health, I will ensure to spawn the player in a safe position, to avoid user frustration.

Data Structures, variables, classes with justifications.

The data structures I am planning to use for my project currently include: arrays(1D and 2D); classes; Dictionaries; lists.

Class diagrams:

Class name:Enemy
Self.x : integer Self.y : integer Self.width : integer Self.height : integer Self.health : integer Self.damage : integer Self.category : string Self.movementx :integer

Self.movementy :integer self.totalHealth : integer Self.facing : boolean
draw(self) move(self)

Class name: Projectile
Self.damage : integer Self.length : integer Self.collided : boolean Self.gravity : integer Self.x : integer Self.y : integer
draw(self) move(self) collisionCheck(self,tiles)

Variable table with justifications

Enemy class variable table

Variable	Data type	Purpose, Justification
x	integer	Stores the horizontal position of the enemy. This is so it can be drawn in the correct place on the screen.
y	integer	Stores the vertical position of the enemy. This so it can be drawn in the correct place on the screen.
width	integer	Stores the width of the enemy, so when creating the rect that will represent the enemy, it can be created with the proper dimensions/ hitbox. This means the user won't get frustrated by inaccurate hitboxes.
height	integer	Stores the height of the enemy, so when creating the rect that will represent the

		enemy, it can be created with the proper dimensions/ hitbox. This means the user won't get frustrated by inaccurate hitboxes.
health	integer	Stores the current health of the enemy, this is so you can determine when to kill the enemy, or maybe even which image to display, e.g. a low health enemy looks more ragged.
damage	integer	Stores the amount of damage a specific enemy can do, so it can be different for each enemy, this allows me to create different enemy variants and makes the game more interesting.
category	string	Stores the enemy class as a string. I could use an integer for this, but then I would have to remember which enemy corresponds to what number, whereas just having”attacker” is easier. This will increase the maintainability of my code.
movementx	integer	Stores the horizontal movement of the enemy at that particular moment, It may change if the enemy for example swaps direction or takes knockback.
movementy	integer	Stores the vertical movement of the enemy. This needs to be an integer as you can't move an enemy by, for example, half a pixel on the screen. This will update if the enemy is falling, as its y velocity would increase due to gravity.
totalHealth	integer	You need to store the total

		health of the enemy just so when deciding what image to display corresponding to its health, you can compare how much health it has lost in comparison to its total. An enemy with 2/4 health would display a very beaten image, but 98/100 would still look normal.
facing	boolean	This is a boolean as it can be set to 2 different values, True or False, in this case that can represent left and right. And therefore you can check what direction the enemy is facing using this attribute.

Projectile class variable table

Variable	Data type	Purpose, justification
damage	integer	To store the damage of a particular projectile. This will be an integer as I don't want things to deal non integer amounts of damage.
length	integer	This will store the size of the projectile, it must be an integer as you cannot draw an image of 0.5 pixels on a screen, it has to be an integer.
collided	Boolean	This is boolean because the projectile will have 2 states, collided and not collided. This allows me to check for collisions and then perform the corresponding actions like deleting a block or dealing damage.
gravity	integer	This attribute is stored so I can change the gravity for different projectiles if I want them to fall down faster(even though objects will accelerate due to gravity at the same

		rate despite their mass). Maybe I will make a projectile that I don't want to be fired very far so I will increase the gravity for it specifically. This will make for more interesting gameplay.
x	integer	This will hold the x position of the projectile. It will be an integer as again, you cannot display something starting on half a pixel. This also allows me to draw the projectile at the correct point on the screen and have information on its whereabouts.
y	integer	Exactly the same as the x variable, except it represents vertical position.

Other Variable Table

Variable	Data Type	Purpose, Justification
screenWidth screenHeight	Integer	Stores the width and height of the screen in pixels. This will allow me to reference it later for calculations, or change the value of the screenWidth everywhere by altering it one place, thus potentially saving lots of time in development.
red green white black	Tuples	Stores the RGB values of basic colours for me to later use to save me from retyping them alot.
runningRight runningLeft crouchingRight crouchingLeft	Arrays	These arrays hold the different images in each respective animation cycle. I am using arrays as it allows me to display different images in the cycle very easily just by for example doing

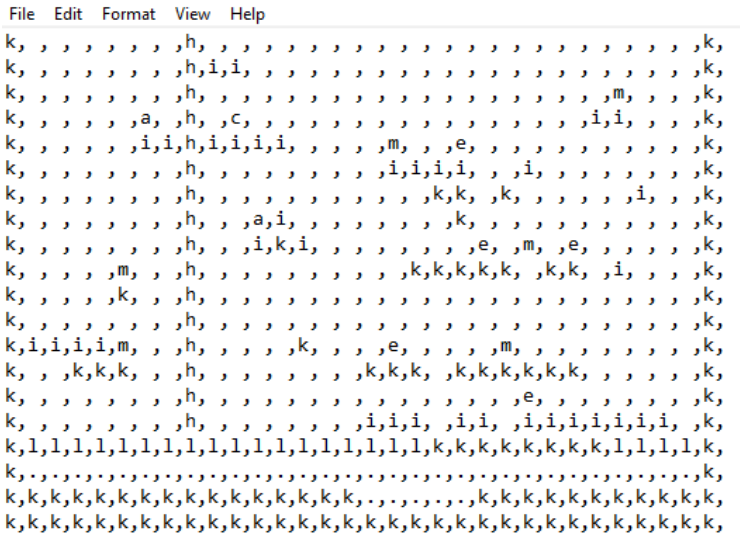
		win.blit(runningRight[2],(player_rect[0],player_rect[1]))
crouching moving_right moving_left climbing isJump isPlayerAttacking isPlayerHit isCollided canClimb	Boolean	These variables will hold either a true or false value, so I can check if the player is doing any of the following and then run the appropriate following code. They are Booleans as they only need two states, as they can only be crouching or not crouching for example.
Player_rect Sword_rect	Instances of the pygame built-in rect class	The purpose of these is to hold the coordinates and corresponding widths and heights of the objects. However it also will create a rect to represent them. I am doing this as it then allows me to use pygames built in rect collision detection methods, so I do not have to invent this code myself - thus saving me development time.
cantCollideList ladders coins enemies projectiles crystals ammoCrates	Arrays	These arrays will hold all the rect objects of the corresponding tile/object . E.g. the crystals array will allow me to access each crystal rect and check it for collisions. This will save time as I can easily check all elements of an array with an iterative process. The cantCollideList is a list of all the rects the player cannot be allowed to collide with(in other words must be able to walk through) like ladders. This is necessary to stop them getting stuck.
level1 level2 level3 level4	Arrays	These arrays will hold the original level details, before anything is added to them. The program will then run

level5		through a level text file and change each element in the 2d array to set up the level details, but to do this an initial array is needed. I have again used arrays as they allow me to go through them in an easy iterative process to set up the levels.
damageCounter coinCounter shotsFiredCounter playerKnockbackCounter playerDealDamageCounter	Integers	These integers are all used to hold counters. These are necessary as the computer is very fast and does many iterations of my code each second. Therefore it may appear that the player will die immediately after touching an enemy, as the computer will think that they have been in contact with the enemy for a large amount of time. Therefore I need the counters to ensure things like the player can only take damage so often. This is important as it will stop the player from becoming frustrated by things like dying instantly, or firing all the ammo with a single shot, and will therefore lead to a more enjoyable game.
playerHealth playerMoney playerAmmo	Integers	These will hold the value of their respective variable names, e.g. playerMoney will hold 5 if the player has collected 5 coins. This is so I can display the correct information in the counters that will be at the top of the screen. They are integers and not real as I have no need for half coins for example. This will make the game simpler and therefore more fun.

Unfortunately constants aren't available in python that I know of, so while many of my variables won't change, for example green will always be (0,255,0), I cannot write
CONSTANT green=(0,255,0)
And save time by allowing the compiler to use immediate addressing. This is why I don't have any "Constants" but rather just variables that don't change, However it is my understanding that as these values don't change anywhere in my program the compiler will save time by treating them as constants. This will increase execution times on my code and make everything look smoother and therefore better for my stakeholders and target audience.

Files

I am going to use simple text files to store my level information, as previously explained in "Explanation of game objects - loading levels...". This is to make it easier to create a variety of unique levels in a way that is easy for me to understand, and therefore utilise.
Each letter in this text file represents a tile (for example in this demonstration h represents ladders, l represents lava). I have done it this way as there are more than enough letters for me to represent all the different in-game tiles I have.



The commas are to separate each individual tile from the one after it, so when I import the file to my code I can use what I have outlined in the pseudocode algorithm section above:

```
DEF FUNCTION importLevel(levelNumber):
    file = open(levelNumber,"r")
    x = 0
    for line in file:
        content = line.split(",")
        for i in range(len(content)-1):
            levelArray[x][i] = content[i]
        x +=1
    return(levelArray)
END FUNCTION
```

And properly split the text file into all the separate elements.

So far this is the encoding for each letter (the letter in the text file and what it will represent)

Letter	Representation
h	ladders
l	Surface lava
.	Under surface lava(i.e no bubbles)
i	Floor
k	Different floor(maybe dirt or something). This is so I can make each level appear different.
c	Crystals
m	Coin
a	Ammo crate

I will have a separate text file for each level, to make it easier to load levels as I will not have to worry about where one level starts and the next begins. This will also make it easier for me to edit levels as I can find the one I want to alter easily.

Input Validation

Input validation is necessary as I cannot allow the user to enter any incorrect keys that may let the program reach an error and crash. I will also have to check for things like my player trying to fire a projectile with no ammo, or leaving the screen.

To stop the player leaving the screen I will just surround it by floor tiles, as seen in the example level above where everything is surrounded by “k”. I am doing this to save me from checking the players movement and if they are close to an edge i.e in danger of falling off the screen.

To stop the player firing with no ammo I will have to check the playerAmmo when they click the mouse, and if it is greater than or equal to one, proceed with the code to fire the projectile. This is necessary as it will stop the player from essentially having an unlimited source of ammo- which will not be enjoyable as it removes an aspect of challenge from the game.

Other forms of input validation that I will utilise will be checking that you cannot move the selection bar beyond the top and bottom options on the menu screen, and if I end up

implementing a melee weapon cooldown then you can only use the weapon once the cooldown has been completed.

As for checking whether incorrect keys are entered, the way the pygame module I am using works is that I get the most currently pressed key every iteration of my code. I will then check the value of this key and if it links to one of my keybinds e.g. “a” for moving left then I will make the program proceed with the left movement code.

Identifying and justifying test data

Key features to test:

1. moving/jumping, basic level
2. Menus
3. ladders/climbing
4. Enemies
5. Health bars and taking damage
6. Level objectives and finishing levels,Coins and point system
7. Attacking,Crouching
8. Importing level details from text files
9. Ranged weapons
10. highscores
11. Sounds effects
12. Instructions

All of these key features must be tested as without them core functionality of the game will be missing. In the tables beneath the “Thing to be tested/inputs” column are the tests that must be completed to ensure that these features function correctly. As such they are ***justified tests***, without them I cannot be sure my game works.

Feature 1: Moving the player

Test Number	Thing to be tested/ inputs	Expected outputs
1	Player moves in the correct direction A,D,SPACE.	A/D - Left and right accordingly. SPACE - Player jumps.
2	Player cannot walk off the screen. Walk to the edge and attempt this.	Player reaches the edge and stops.
3	Correct animation is displayed. Inputs are A,D,S.	A/D - the run animation(Left/right) S - the crouching animation.

Feature 2: Menus

Test Number	Thing to be tested/ inputs	Expected outputs.

1	Moving between options on the menu screen. Inputs: up/down keys, enter for select.	A bar should move to show which option is currently selected. When enter is pressed it should move to the corresponding screen.
2	You cannot select anything other than the available options(cant press up key and go above the top option)	Bar showing selection will not move up/down beyond the top/bottom options despite pressing the up/down key.
3	Any other button will not do anything, e.g. F key, BACKSPACE key.	No output.

Feature 3: Using ladders

Test Number	Things to be tested/ inputs	Expected outputs
1	Players can walk through ladders if they don't want to use them. Inputs: movement keys.	Player reaches the ladder and walks through it unimpeded.
2	Player can climb and descend ladders. Inputs: W (player can descend if not holding W) .	If player is at same co-ords as a ladder and presses W, they should rise, if then lets go of W, they should fall.
3	Player stops climbing when they reach the top of the ladder.	When player reaches the top of the ladder they fall off.

Feature 4: Enemies spawning and moving

Test Number	Thing to be tested/ inputs	Expected outputs
1	Enemies spawn in correct location.	For each level the enemies spawn corresponding to their locations in the text files storing the levels.
2	Enemies traverse their environment.	Enemies walk left and right.
3	Enemies do not walk off the respective platforms.	Enemies reach the edge of their platforms and face the other way.

Feature 5 : Health bars and taking damage

Test Number	Thing to be tested/ inputs	Expected outputs
1	Taking damage- use movement keys to collide with an enemy or lava block.	Health bar will decrease.
2	Health bars are capped at a maximum and minimum value. Walk into lava using movement keys and stay there.	Health value does not go below 0. Bar doesn’t become drawn in the negative direction(as health is proportional to the width of the bar in pixels.)

Feature 6: Level objectives and finishing levels

Test Number	Things to be tested/ inputs	Expected outputs
1	When the player touches crystals they disappear.	When rect objects collide, the coin/ammocrate/crystal disappears.
2	The corresponding value increases	The counter which will be displayed at the top of the screen is incremented.
3	As well as disappearing they get deleted from level to stop them from being reused whilst invisible.	When the player moves over the spot the object once was in nothing happens(to the counters)
4	Level will only end when all crystals are collected	Collect crystals one by one and only when the last one is collected will the game move to the next level.

Feature 7: Attacking, crouching

Test Number	Things to be tested/ inputs	Expected outputs
1	Dealing damage- use the sword keybind and walk into an enemy with it.	Enemy health will decrease, they will eventually change images to their corresponding health percentages, and eventually die(disappear from screen). Enemy kill count should increment if this happens.
2	Applying appropriate amounts of damage.	For a collision with an enemy, the taking damage counter should stop the player from taking too much damage. Health bar should only go down around a tenth(this value is subject to change as I see fit). Enemy should also not die immediately.
3	Player can only un-crouch at appropriate times	When crouch key is released if under one block high gaps the player wont uncrouch

Feature 8 : Importing level details

Test Number	Things to be tested/ inputs	Expected outputs
1	The correct level loads	The levels are loaded and played in the intended order

Feature 9: Ranged weapons

Test Number	Things to be tested/ inputs	Expected outputs
1	You cannot use more ammo than you have. Inputs: mouse buttons.	You cannot fire a projectile when the ammo counter is 0. Ammo counter will

		not go below zero.
2	The path the projectiles follow	The players projectile follow a realistic path
3	Destroying blocks/damaging enemies	Destroys the blocks or damages the enemies depending on what it hits

Feature 10: Highscores

Test Number	Things to be tested/ inputs	Expected outputs
1	Highscores are displayed	Top 5 scores shown on screen
2	Displayed in correct order	The scores are sorted numerically, largest first.

Feature 11:Sound effects

Test Number	Things to be tested/ inputs	Expected outputs
1	Sound effects play	Sound effects at a suitable volume
2	Correct sound effects play	The right sound effect for the scenario is played

Feature 12 : instructions

Test Number	Things to be tested/ inputs	Expected outputs
1	Instructions are shown	When instruction option is selected they are shown
2	You can leave the instruction screen	When the exit key is pressed player returns to the main menu

Further data for post development with justifications

Post Development Test	Testing to be done	Justification
Level difficulty increases	More enemies each level? Stronger/ better positioned enemies each level? Do the levels seem a suitable difficulty- get stakeholders to test this for me.	The levels have to be at a suitable difficulty in order to be fun, too easy and there is no challenge, too hard and people will get fed.
Easy to use the projectiles	Does the projectile path feel intuitive? Is acceleration due to gravity	If the player finds it difficult to aim then they will not use this feature, which will make the

	too great?	game less interesting.
Weapons are adequately balanced.	Does the sword/ projectile deal enough damage? Are enemies too hard to kill?	I don't want the player to feel that the health the enemies have is too unfair. Therefore I will make sure the weapons don't deal too little damage.
Hitboxes are an adequate size.	Do the enemies take damage when the sword/ projectile image appears to collide, not when the underlying rect objects actually collide.	The hitbox has to fit the image of the enemies, not the enemies themselves, as the images are what the player uses to identify targets. Otherwise the player may feel cheated which will not make for a fun game.

Checking the final designs meet stakeholder requirements:

At this point, my ideas for the game are taking form, and before I begin developing the game I would like to check that my design ideas meet the stakeholders expectations.
As such I sent them the following email:

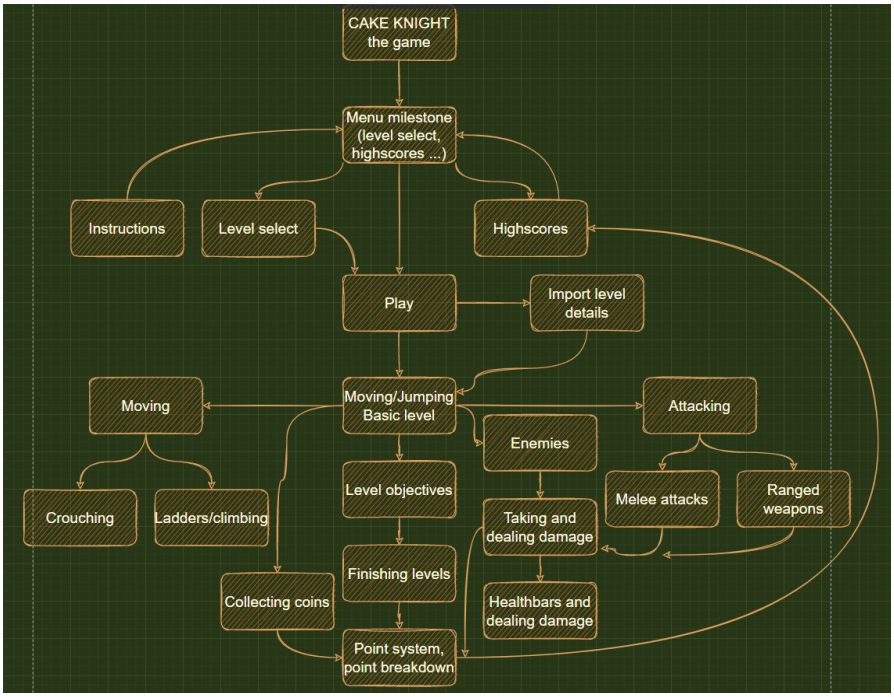
> “
>
> Dear Joe & Charlotte,
> I am in the process of finalising my design for the game, and at this point I would like to give you a summary of my ideas, and get some feedback from you guys. My ideas include:
> - A level objective that includes collecting items hidden around each level. At this moment in time I want to make this a crystal
> - Points that can be collected with some sort of coin or by defeating enemies
> - A range of 5 levels of increasing difficulty
> - A menu with a high score, level select and help option
> - Ladders and some sort of power up like a ranged weapon, of which there will be limited ammo to use hidden around the map
>
> I look forward to hearing from you shortly.
> ”

These are their responses:
Joe:”These all sound like good ideas. I particularly like the powerups, that sounds fun. Please make sure though that the game doesn't become too complicated, as I just want a simple fun game.”

Charlotte:"I like the idea of the level objectives being hidden throughout the map, this sounds like a good challenge. The highscore idea I like as it will give me something to work towards. Can you make the coins animated, I think this would make everything look nicer, and the game aesthetics are important to me"

What I have learned from this: my current ideas will be acceptable to my stakeholders and therefore also my target audience. However I must ensure that I animate the coins and that I do not make the game too complicated.

Developing coded solution



To the left is the representation of how my algorithms link created in the design stage.

This is so I can reference it throughout the development process, and so I know the order in which to code my program iterations.

Documenting Milestones(with user feedback, testing & reflections for each milestone):

Outline of my iterative approach to coding the solution

For each milestone I have the following sections:

1. **Objective** (What I aim to achieve by the end of the iterative process)
2. **Code explanation** (Commented code with written explanations and justifications)
3. **Testing and errors** (Testing carried out, errors encountered, fixes implemented with justifications)
4. **User feedback** (input from my test user for refining each milestone)
5. **Reflections** (success criteria completed, thoughts on any potential further optimisation)

Reasons for use of milestones:

These key milestones are my problem decomposed into smaller subproblems. I have done this because:

- I will be able to check my progress easily.
  Each one represents a key piece of functionality that must be completed. Therefore by completing all of these milestones in turn, I will be have a complete game, and so throughout the development process I know exactly where I am at in the development stage.
- I can independently test each milestone to check that it works.
  By completing each milestone in turn, and checking the functionality works with tables created in the "Identifying and justifying test data" segment of the design section. I will know any errors will be due to the milestone being currently being implemented, allowing me to isolate and fix bugs easier.
- A larger, more complex problem becomes easier to solve.
  By breaking down the problem (decomposition mentioned in Analysis section) each milestone now represents one task, and is now easier to program because of this

Milestone Checklist

- ☐ moving/jumping, basic level
- ☐ Menus
- ☐ ladders/climbing
- ☐ Enemies
- ☐ Health bars and taking damage
- ☐ Level objectives and finishing levels,Coins and point system
- ☐ Attacking,Crouching
- ☐ Importing level details from text files
- ☐ Ranged weapons
- ☐ highscores
- ☐ Sounds effects
- ☐ Instructions
- ☐ Polishing features, sword animation, shield animation

Success criteria checklist

These success criteria are as seen in the "analysis" section. For each documented milestone, the criteria completed are included in the *reflections* section.
✔️- means successfully implemented
❌- means abandoned the criteria- no longer necessary or should not be implemented

Success criteria number	Completed?	Success criteria number	Completed?
1	✔️	10	✔️
2	✔️	11	✔️
3	✔️	12	✔️/❌
4	✔️	13	✔️
5	✔️	14	✔️
6	✔️	15	✔️
7	✔️	16	✔️
8	✔️	17	✔️
9	✔️	18	❌
19	This cannot be determined until my stakeholder review section in the evaluation		

Criteria 18 is no longer deemed necessary and so has been marked❌. Sound toggles are not needed as there is no background music (explained in the sound effects section and seen in criteria 12 marked as❌) so the only toggleable option is sound effects, which you can turn off with volume control if necessary.

Below is the documentation for each of my main milestones. I have also recorded the number of attempts each of these sub problems have taken me to complete,(the number of coding iterations I spent on each milestone).

Moving/jumping - basic level

Objective:

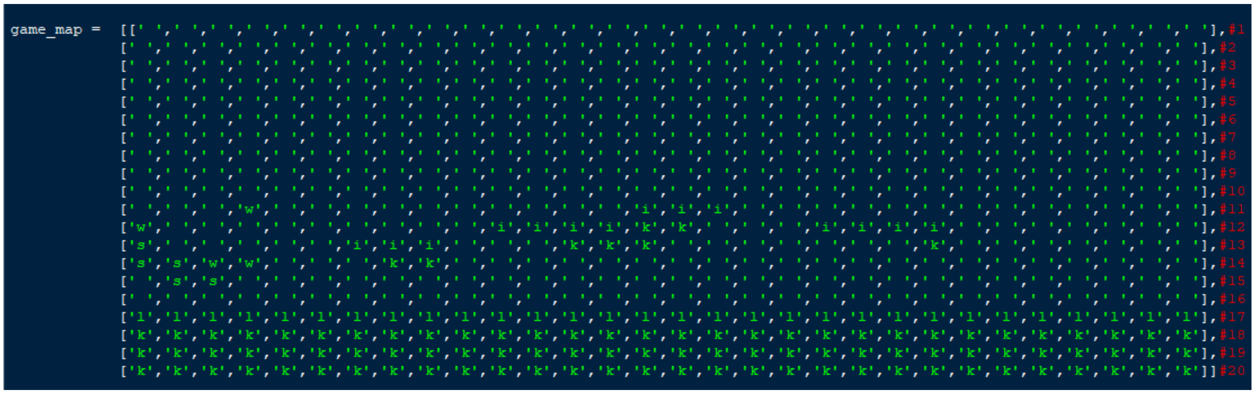
- To have a basic level
- To be able to have a player that will move on screen
- Player will not fall through the floor.

Code explanation:

Drawing the level(floors etc.)

To draw/create a basic level, I need a way of storing the level details. In the future a text file would be best but at the moment I don't want to overcomplicate my milestone so I will store it in

an array. Then I will go through the array and for each element create the corresponding rect object that I can use for detecting collisions,add this to an array tile_rects, and stick the corresponding image over it to make everything look nice.(I've taken a print screen as otherwise the array is too long to fit nicely)



```
game_map = [[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#1
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#2
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#3
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#4
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#5
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#6
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#7
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#8
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#9
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#10
            [' ',' ',' ','w',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','i','i','i',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#11
            ['w',' ',' ',' ',' ',' ',' ',' ',' ',' ','i','i','i','i','k','k',' ',' ',' ','i','i','i','i',' ',' ',' ',' ',' ',' ',' '],#12
            ['s',' ',' ',' ',' ',' ','i','i','i',' ',' ',' ','k','k','k',' ',' ',' ',' ',' ',' ',' ','k',' ',' ',' ',' ',' ',' ',' '],#13
            ['s','s','w','w',' ',' ',' ','k','k',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#14
            [' ','s','s',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#15
            [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],#16
            ['l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l','l'],#17
            ['k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k'],#18
            ['k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k'],#19
            ['k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k','k']]#20
```

```
tile_rects = []#List of the tiles to create for the level
  y =0
  for row in game_map:
      x = 0
      for tile in row:
          if tile == 'w':# if the letter is w
              win.blit(grass, (x * TILE_SIZEX, y * TILE_SIZEY))#add a grass image to the screen
          if tile == 's':
              win.blit(dirt, (x * TILE_SIZEX, y * TILE_SIZEY))
            if tile == 'i':
                win.blit(grassCobble, (x * TILE_SIZEX, y * TILE_SIZEY))
            if tile == 'k':
                win.blit(cobble, (x * TILE_SIZEX, y * TILE_SIZEY))
            if tile == 'l':
                win.blit(lava, (x * TILE_SIZEX, y * TILE_SIZEY))
            if tile != ' ':# if the letter does not represent air
                tile_rects.append(pygame.Rect(x * TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))# add the tile to an array for use in checking collisions
            x += 1
        y += 1
```

Moving the player

From **lines 2 to 14** handles the walking animation of the character. The runningLeft/runningRight arrays each hold three images of the character, in time I may add more in order to make the animation seem more fluid. As the player moves a counter increments which will swap the sprite currently being used in the animation.
The players x and y velocities at any one time are stored in the list player_movement, it is not a tuple as then I would not be able to change the values.

```
1.    player_movement = [0,0]
2.    if moving_right:
3.        if walkCycle>26:#If the animation cycle is complete
4.            walkCycle = 0#rest it
5.        walkCycle +=1
6.        player_movement[0]+=5#add 5 to the players movement
   variables
7.        playerImage = runningRight[(walkCycle)//9-1]#display
   appropriate image
8.  #the moving left code functions the same as the moving right
9.    if moving_left:
10.         if walkCycle>26:
11.             walkCycle = 0
12.         walkCycle+=1
13.         player_movement[0]-=5
14.         playerImage = runningLeft[(walkCycle)//9-1]
15.
16.     player_movement[1] += player_y_vel# add gravity to the y
   movements
17.     player_y_vel += 0.2
18.     if player_y_vel > 7:#cap the movement due to gravity at a
   certain level
19.         player_y_vel = 7
20.
21.     player_rect, collisions = move(player_rect,
   player_movement, tile_rects)
```

Jumps are achieved by checking if the player presses a space bar, and that they have been touching the ground very recently (this allows you to jump if you have just walked off the edge, a common feature in most games)

```
1. if collisions['bottom']: # if the player is touching ground
2.        player_y_vel = 0 # set y velocity to zero
3.        air_timer = 0
4.    else:
5.        air_timer += 1
```

```
1. if event.key == K_SPACE: # if space is pressed
2.                 if air_timer < 6: # if recently touched ground
3.                     player_y_vel = -7#make the player jump
```

The player's y velocity is set to a negative 7 as the top of the screen is zero, and below this is positive y coordinates(opposite to normal graphs) so to make the player move up I have to subtract from the y coordinate.

Making sure the player won't fall through the floor

The contents of this list are passed to the move function detailed below on **line 21**. This is to ensure that we only allow the player to move if they should be able to.

```
1. def move(rect, movement, tiles):
2.     collision_types = {'top': False, 'bottom': False, 'right':
   False, 'left': False}#stores dictionary of possible collision
   types
3.     rect.x += movement[0]#moves the player in the x direction
4.     hit_list = collision_test(rect, tiles)
5.     for tile in hit_list:
6.         if movement[0] > 0:#tests for collisions
7.             rect.right = tile.left
8.             collision_types['right'] = True
9.         elif movement[0] < 0:
10.                rect.left = tile.right
11.                collision_types['left'] = True
12.        rect.y += movement[1]#moves the player in the y direction
13.        hit_list = collision_test(rect, tiles)
14.        for tile in hit_list:
15.            if movement[1] > 0:#tests for collisions
16.                rect.bottom = tile.top
17.                collision_types['bottom'] = True
18.            elif movement[1] < 0:
19.                rect.top = tile.bottom
20.                collision_types['top'] = True
21.        return rect, collision_types#return any collisions that
   will occur in the form of the dictionary
```

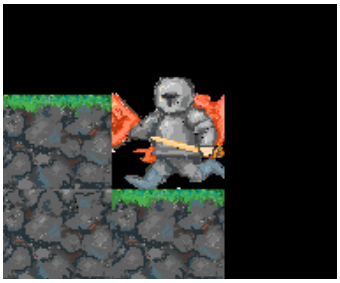
An array hit_list is created with a list of all the collisions the player has at that moment in time. For each one of these collisions, I check what direction the player was moving in when this collision occurred. If they were moving to the right, it means they have collided with a block to the right. **Line 7** will move the player back out of the block(the user will see this as the player never having entered the block yet as the screen hasn't been updated yet.) A collision right will now be recorded by changing the value of "right" in the dictionary.

If there is a bottom collision, i.e the player is walking on something, their y velocity gets reset, (it is constantly being added to to simulate gravity.)

```
1.    player_movement[1] += player_y_vel #moves the player
2.    player_y_vel += 0.2 #adds the gravity to the y vel
3.    if player_y_vel > 7: #caps the gravity at a certain value
4.        player_y_vel = 7
```

Testing and errors:

I had to test that the code to stop the player walking through walls worked, and it has. It took a while to tweak the values for the players movement speeds that I wanted, and the jump height(this is dependant on the size of that negative value I assigned to the y velocity earlier when the space bar is hit) because I didn't want the player to be able to jump too high.



In this image, despite the key for moving left being held, the player is prevented from walking into the wall, they are also not falling through the floor, so the code is a success.

Below are the lines of code I altered to change the movement speeds(this is lines 2,3,4 above):

```
1. player_y_vel += 0.4
2. if player_y_vel > 10:
3.       player_y_vel = 10
```

This has now fixed the problem of slightly off jump heights and odd paths due to gravity.

Test Number	Thing to be tested/ inputs	Expected outputs	Success?
1	Player moves in the correct direction A,D,SPACE.	A/D - Left and right accordingly. SPACE - Player jumps.	PASS
2	Player cannot walk off the screen. Walk to the edge and attempt this.	Player reaches the edge and stops.	PASS
3	Correct animation is displayed. Inputs are A,D,S.	A/D - the run animation(Left/right) S - the crouching animation.	PASS

User feedback:
My user helped me to find a value for the jump height that felt natural and wouldn't allow the player to navigate too easily and thus get rid of an aspect of challenge. They liked the anti collision and said it looked good that the player couldn't walk into walls. Therefore I know they are happy with this milestone, and I can move on to the next one.

Reflections:
If the jump is too high or low I can always tweak this later, and may have to depending on how level designs vary, if I want the player to be able to make bigger/ smaller jumps.
This milestone took me 1 main attempt to complete, as there were no errors apart from tweaking values.

This is success criteria 1-**completed**

1	Must have WASD for movement, with the addition of spacebar for jump(as w will be for ladders.)	Many people from the target audience have expressed their wish for these keys to control the movement of the player, due to their common use in pc games

Menus
Objective:

- Have a menu screen that allows interaction, will let the player choose to play, which should start from level one, or a level select or highscore option, (highscore will be completed in a later milestone.)

Code Explanation:
Creating the menu screen
The image of the menu screen is loaded onto the screen in **line1**. If a key press is detected, **line2**, I check if the user should be able to move the yellow selection bar up/down (they may already be at the top/bottom of the list and therefore should be able to move no further), and if they can move I will move the bar to show a different option selected.When enter is pressed the final choice is finalised, and you will exit the while loop for the menu screen and move onto the corresponding screen, either level select, play or highscore (which will be completed later, as I only need the basic functionality for this stage)

```
1.    while menu:
2.        win.blit(splashScreenImg,(0,0))#draw the menu screen
3.        for event in pygame.event.get():
4.            if event.type == KEYDOWN:#can press the arrow keys to
   move between options
5.                if event.key == K_DOWN:#if arrow key pressed
6.                    if choice < len(options)-1:# restricts
   choices
7.                        choice +=1
```

```
8.                          print(options[choice]) #prints choice for
   ease of debugging
9.                  elif event.key ==K_UP:#if up key
10.                         if choice >0: #if can move up
11.                             choice -=1 #move up
12.                             print(options[choice])
13.                     elif event.key == K_RETURN: # when enter is
   pressed option is finalised
14.                         selected = True #finalise choice
15.                         menu = False #escape from the menu loop
16.                         print("Final choice",options[choice])
17.             LinePos1 =(100,(600+choice*200))
18.             LinePos2 =(700,(600+choice*200))
19.
   pygame.draw.line(win,(255,215,0),(LinePos1),(LinePos2),10)#surfa
   ce,colour,start,end,width
20.
21.             pygame.display.update() # updates the display
22.
23.         if options[choice] == "play":          #
24.             game = True                        #
25.         elif options[choice]=="levelSelect": #sets the variables
   for each while loop
26.             levelSelect = True                 #depending on what
   was picked in the splash screen menu
27.         elif options[choice] == "highScores":
28.             highScores = True
```

The selection bar is necessary for the player to see what they have currently selected.

Testing and errors:

I had to test that the yellow option select bar would appear at the correct place on the screen for each option, it took a while to decide on the values seen in **lines 17 and 18** which decide the position of the line. I settled on these numbers as they best centred the line over the options.
I also had to test that each option took me to the required screen (however highscores was just a blank screen as I have not created this feature yet).

The yellow selection bar fits nicely under the options available, these are some images of it in its different positions.

Test Number	Thing to be tested/ inputs	Expected outputs.	Success?
1	Moving between options on the menu screen. Inputs: up/down keys, enter for select.	A bar should move to show which option is currently selected. When enter is pressed it should move to the corresponding screen.	PASS
2	You cannot select anything other than the available options(cant press up key and go above the top option)	Bar showing selection will not move up/down beyond the top/bottom options despite pressing the up/down key.	PASS
3	Any other button will not do anything, e.g. F key, BACKSPACE key.	No output.	PASS

User Feedback:

The yellow bar is liked, some people said they would prefer to be able to select options with the mouse that will highlight a box around an option when hovering over it, having considered this I will not implement yet if at all, as it will require more time and more complicated code and there are more basic functionalities to be created first. They thought it looked odd that the yellow bar does not fit each word, and is slightly longer than necessary for play and level select. They

thought the start menu could use some more colours, so I adjusted it to look like this.

Reflection:

If there is time at the end of development I can always change the way of selecting options to mouse input. However this won't be necessary as the functionality is there already, it may add to a sense of fluidity.

This milestone took 1 attempt to complete. The code was a success and little testing was needed to identify the values needed to store the locations of the selection bars.

This is success criteria 10- **completed**

10	A menu with highscores, level select and play option.	Player should be able to see their high scores, or choose which level they want to play based off of their difficulties.

Ladders/climbing

Objective:

- To be able to have ladders in my level that will allow the player to climb up and walk past and not use if wanted.

Code Explanation:

Creating the ladders

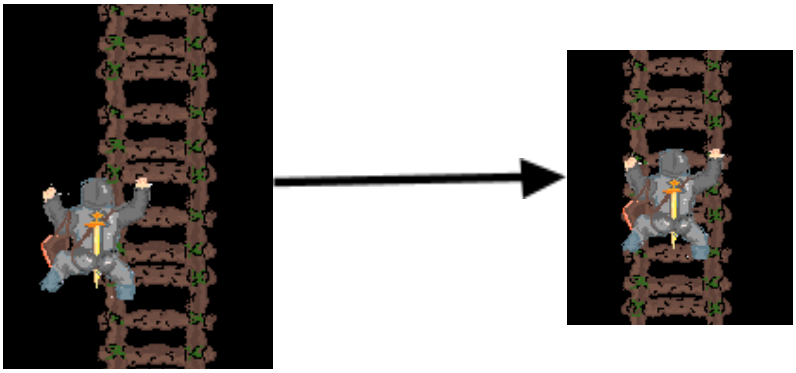
If the player is climbing(The user is holding down the climb key), then the algorithm will check if the player is able to climb something(canClimb). I.e if they are touching a ladder(**line 7**) the boolean variable canClimb will be set to True. It is boolean as canClimb should only ever be one of two states, True or False. On **line 9** the loop oneTime is necessary to make the animation smooth, I had to make sure the player was fully centred over the ladder when climbing so they didn't look like they were hanging off of it. To do this I had to set their coordinates to that of the part of the ladder they were trying to climb. This had to be done only once otherwise they would never go anywhere if they kept getting reset.
Then if canClimb = False it resets the climbing and oneTime variables, and if not it proceeds to let the player climb. This is achieved by subtracting 3 from the y coordinate(upwards has a decreasing y coordinate) each iteration of the loop, and changing their animation too to make the game more polished.

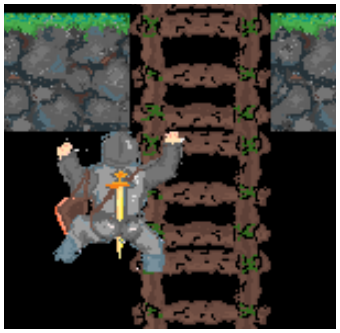
```
1. if climbing:# if the player is climbing
2. player_y_vel = 0#makes it so the player wont fall off
3. moving_right = False#stops the player moving off the sides
   accidentally
4. moving_left = False
5. canClimb = False
6. for ladder in ladders:
7.         if player_rect.colliderect(ladder):#is the player
   touching ladders?
8.                 canClimb = True # sets climbing to true
9.                 if oneTime:
10.                        player_rect[0]=ladder.x
11.                        player_rect[1]=ladder.y#only resets the
   player co-ords once
12.                        oneTime = False
13.   if canClimb == False: #if player isn't next to ladders,
   climbing is false
14.           climbing = False
15.           oneTime = True
16.
17.   else:
18.
19.            if climbingCycle>23:#if the player can climb, then
   they climb
20.                    climbingCycle = 0
21.            climbingCycle +=1 # sets the animation for climbing
22.            playerImage = climbingImages[(climbingCycle)//12-1]
23.            player_rect[1]-=3 #makes the player climb
```

Testing and errors: ***(inc. description of error encountered, fixing and justification of fix)***
I had to make sure that the climbing feature worked, as often with big additions like this things go wrong. In testing before the addition of the code to centre the player on the ladder I found

that they would glitch through the ground. This is because when climbing the ladder, if you are not centred on it then you will collide with blocks that are next to the ladder. If this happens the collision detection code will try and reset your position to the top of the block, which makes it appear as though you teleport through it. It is also difficult to stop climbing the ladders and walk onto a platform as you have to let go of the climbing key and then move. I can write code to centre the player on the ladder, and then from there allow left and right movement on the ladder, this will make it easy to get off the ladder. However I will have to include code to detect the moment at which the player has left the ladder and then set the climbing boolean to false. I will also then need to find another solution to the teleporting through blocks problem.

By changing the code on **line 23** to make use of the player_movement command to say player_movement[1] -=5 this way I can use the collision detection and correction feature in my move function which will reset the player if they try to climb up through a block.. Everything works fine now, and it is easy to go on and off the ladder, you can even climb up the sides of a ladder as shown below, and you won't go through the underside of blocks anymore.





This is proof that the player can no longer go through the underside of the blocks

Test Number	Things to be tested/ inputs	Expected outputs	Success?
1	Players can walk through ladders if they don't want to use them. Inputs: movement keys.	Player reaches the ladder and walks through it unimpeded.	PASS
2	Player can climb and descend ladders. Inputs:	If player is at same co-ords as a ladder and presses W, they should	PASS

	W (player can descend if not holding W) .	rise, if then lets go of W, they should fall.	
3	Player stops climbing when they reach the top of the ladder.	When player reaches the top of the ladder they fall off.	PASS

User feedback:
Joe said it was initially hard to climb off the ladder as you cant move left and right whilst on it. He said he did manage to get used to letting go of the climb ladder button(W) to then be able to move off of the ladder. After fixing the ladders he said it worked great.

Reflections:
It was good that I took the time to fix the ladder mechanics, as my test user said that it made the gameplay much smoother.
This milestone took me 2 main attempts to complete, as many faults were detected with my initial algorithms as outlined above in my testing and errors section and another iteration was needed to fix them and test my solutions.

This is success criteria 15-**completed**

15	Include ladders for vertical traversals too big for the jump feature.	Means level design has less constraints, making for better levels. Also adds gameplay possibilities like half climbing a ladder to avoid an enemy.

Enemies:

Objective:
- To have enemies that will move around on their designated platforms with their own health attributes.
- The basic enemy will also be able to attack the player by moving into them.
- I will also add floating enemies that sit on a platform and fire ranged weapons at the enemy.

Code Explanation:
Creating the enemy class
Lines 3 to 15 are the attributes for my enemy class, allowing me to change their widths and heights if I want bigger enemies, they have damage,health and facing attributes, the facing one is to allow me to know what direction the enemy was last moving in so I can have left and right movement animations in the future if I have the time.

Lines 16 onwards are the methods for my enemy class. The draw() method is responsible for drawing and moving the rect object that represents the enemy, which will also allow me to use pygame rect collision detection. It also is responsible for putting the picture of the enemy sprite on top of this rect.
The move() method makes the enemy move either left and right until it bumps into a block or it is about to fall of the edge, and then reverses direction. **Lines 24-33** check if the enemy has been in the air for a certain amount of time(i.e in the process of falling off the edge) and if they have it will put them back on the platform and move them in the other direction. This is necessary as with my skill set I don't have a way to check if the enemy is about to fall off the edge, I have to wait until they are doing it so I can detect it and fix it.

```
1. class Enemy():
2.     def
   __init__(self,x,y,width,height,health,damage,category,movementx
   =3,movementy=0,facing=1,counter=0):#default movement is 3 to the
   right
3.         self.x = x # intialises the instances of the class
4.         self.y = y
5.         self.width = width #all the attributes for the class
6.         self.height = height
7.         self.health = health
8.         self.damage = damage # e.g. damage attribute for enemies
9.         self.category = category
10.        self.movementx = movementx
11.        self.movementy =  movementy
12.        self.facing = facing
13.        self.counter = counter
14.
15.        self.rect =
   pygame.Rect(self.x,self.y,self.width,self.height)
16.     def draw(self):# method for enemy to draw itself
17.         pygame.draw.rect(win,red,self.rect,-1)
18.         win.blit(cakeEnemy,(self.rect[0],self.rect[1]))
19.     def move(self):
20.         self.rect, collisionsEnemy = move(self.rect,
   (self.movementx,self.movementy), tile_rects)#updates the enemy
   rect, retrieves a list of things it collides with
21.         if collisionsEnemy["bottom"]:
22.             self.movementy = 0
23.             self.counter = 0
24.         if  not (collisionsEnemy["bottom"]):  #if not standing
   on block after movement, undoes the movement and turns the enemy
   around
25.             self.counter +=1
26.             if self.counter ==3:# if fallen off the edge fix
   this
```

```
27.                 self.rect[1]-= 1.2
28.                 self.rect[0]-= self.movementx
29.                 self.movementx = self.movementx*-1
30.             if self.movementx>0: #resets the facing variable to
    match the way the enemy is now moving
31.                 self.facing = 1
32.             else:
33.                 self.facing = -1
34.         if (collisionsEnemy["right"]) or
    (collisionsEnemy["left"]):#if hits something to the right turns
    around, no need to reverse movement, this for horizontals is done
    within move function
35.             self.movementx=self.movementx*-1
36.             self.facing = self.facing * -1
```

Basic enemies attacking the player by moving into them.

This code loops through all the enemies in the list holding all of the instances of the enemy class. With each of these it then calls the draw and move methods.
On **line 4** it checks to see if the enemy collides with the player and if it does and the damage counter is equal to zero(This is to avoid the player taking damage too quickly, as the computer will do many cycles of my code per second so I need a counter to limit it). If these conditions are met then the player will lose health. **Lines 12 and 13** stops the player health from going below zero as this would interfere with my design for the health bar (where the health controls a rectangle with width = to the health value and r value in rgb equal to the health value)

```
1.         for enemy in enemies:# draw and move all the enemies
2.             enemy.draw()
3.             enemy.move()
4.             if enemy.rect.colliderect(player_rect) and
    damageCounter == 0:# if player touches enemy
5.                 playerHealth -= 25#damage the player
6.             enemy.movementy+=0.4#handles gravity on the enemies
7.             if enemy.movementy>10:
8.                 enemy.movementy=10
9.         damageCounter +=1#damage counter so the player can't take
    damage too quickly.
10.         if damageCounter ==15: #resets damage counter
11.             damageCounter = 0
12.         if playerHealth<0: #stops player health going subzero
13.             playerHealth = 0
```

Creating the ranged enemies

The ranged enemies have an extra method they can use called attack, this will create an instance of a new orb class(orb is the weapon they fire) which I have included below.
This orb has its numerous attributes allowing me to create ones of different dimensions and with different damages if I want to. The draw() method is the same as most of my classes, it draws the underlying rect and then sticks the image of the sprite on top.
The move() method takes the player_rect as an argument(this has the player coordinates in it). The deltax and deltay on **lines 21 and 22** are the amount of x and y pixels separating the player and the orbs starting position. From this I work out the total distance using pythagoras. Then I calculate the vectors in the x and y direction using ratios to ensure that the overall speed is constant. Then I move the orb by these x and y velocities on **lines 33 and 34**

```
1. class Orb():
2.     def
   __init__(self,length,enemyx,enemyy,player_rect,damage,vel,image
   ,collisionTile = " ", collisionPlayer= " "):
3.         self.damage = damage#initialise attributes for the class
4.         self.image = image
5.         self.length = length
6.         self.collisionTile = collisionTile
7.         self.collisionPlayer = collisionPlayer
8.         self.x = enemyx # sets x co-ord attribute
9.         self.y = enemyy # same is done for all the attributes
10.        self.vel = vel
11.        self.startx = enemyx
12.        self.starty = enemyy
13.        self.calculateVectorOneTime = True
14.      def draw(self):#method to draw the orbs onscreen
15.          self.rect =
   pygame.draw.rect(win,red,(self.x,self.y,self.length,self.length
   ))
16.          win.blit(self.image,(self.x,self.y))
17.
18.      def move(self,player_rect):#method to move the orbs
   onscreen
19.
20.
21.          deltax = self.startx -
   (player_rect[0]+player_rect[2]//2)
22.          deltay = (player_rect[1]+player_rect[3]//2) -
   self.starty  #calculates the difference in the players position
   and the orbs starting position
23.          displacement = (deltax**2 + deltay**2)**0.5
24.
25.          self.velx = (deltax/displacement) * self.vel
26.          self.vely = (deltay/displacement) * self.vel
```

```
27.
28.             if self.startx>player_rect[0] and self.velx>0:
    #makes sure that the velocities have the correct direction
29.                 self.velx = self.velx*-1
30.             if self.starty>player_rect[1] and self.vely>0:
31.                 self.vely = self.vely*-1
32.
33.             self.x += self.velx # moves the orb
34.             self.y += self.vely
```

*Testing and errors:**(inc. description of error encountered, fixing and justification of fix)***

I had to make sure that the values on **lines 26 - 28** on the move method in my enemy class worked. It took several iterations of the enemies not being able to make it back onto the platform, or moving them too high up to find the right values. I also had to play around with the values for the taking damage counter, as too little would be unfair, and too much would allow the player to stand by the enemy which I don't want yet.

```
for orb in orbs:
    orb.move(player_rect)
    orb.draw()
    if orb.x>1920 or orb.x<0 or orb.y>1080 or orb.y<0:
        orbs.pop(orbs.index(orb))
    if player_rect.colliderect(orb.rect):
        playerHealth -= orb.damage
        orbs.pop(orbs.index(orb))
    if shield.colliderect(orb.rect):
        try:
            orbs.pop(orbs.index(orb))
        except:
            pass
```

Using try and except to fix the problem(fixed code on the left)

When testing I encountered a problem in that if the orb hit the player and a shield at the same time the program attempts to delete the orb twice as can be seen in this code. This throws up an error as you cannot delete something that has already been deleted. To prevent the program from crashing when this happens I researched the try: and except: features in python and used these to try and delete the orb if possible, but if not then don't crash over it.

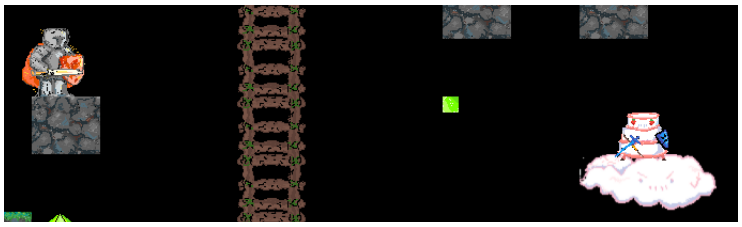


Image of the ranged enemy on the cloud firing an orb. From this I can see that the method to calculate the vectors that moves the orbs to the player works. I also moved the player to different positions around the ranged enemy and it still worked.

Test Number	Thing to be tested/ inputs	Expected outputs	Success?
1	Enemies spawn in the correct location.	For each level the enemies spawn corresponding to their locations in the text files storing the levels.	PASS
2	Enemies traverse their environment.	Enemies walk left and right.	PASS
3	Enemies do not walk off the respective platforms.	Enemies reach the edge of their platforms and face the other way.	PASS

User Feedback:

Charlotte said that she likes the fact that there is more than one type of enemy, but she says to make sure there is a good balance of each type on each level, and not to include many more types as she doesn't want the game to become too hectic. I will bear this in mind for future developments.

Reflections:

These values for the counters can always be changed later in fine tuning and polishing the game. I won't be adding more enemy classes to the game due to time constraints, my user also said that he would prefer that I don't to avoid there becoming too much information to learn to play the game.
This milestone took 3 coding attempts to complete, one for the initial algorithm, another to fix the errors outlined in my testing and errors section and one to find desirable values for my constants.

This is success criteria 4-**completed**

4	At least two different types of enemy.	In my research most platformers have a wide variety of enemies, making for more interesting features and play styles, I want to incorporate this into my game so it can be as fun as possible.

Health bars and taking damage:

Objective:

- To take damage from things like lava and enemies.
- To have a health bar that goes down with damage and progressively changes colour (gets redder) as health goes down.

Code explanation:

Taking damage from things like lava and enemies

I have added an array called damage blocks, this stores the rects of all the blocks which will allow the player to take damage. Then I go through this array and for each block check if the player is in contact with it and if they are then again according to a timer like in the enemies milestone above, I let the player take damage.

```
1.
2.       for block in damageBlocks: #for all of the blocks that
   damage the player to touch
3.           if player_rect.colliderect(block) and healthTimer ==
   0:
4.               playerHealth -=10#if player touches the block they
   lose health
5.               healthTimer +=1
6.               if playerHealth<0:
7.                   playerHealth = 0
8.       if healthTimer>0:  #increments and resets the health timer
   so damage can be dealt at an acceptable rate
9.           healthTimer +=1
10.          if healthTimer>5:
11.             healthTimer = 0
12.
```

Health bar that goes down with damage and gets redder

I have made a function to draw a rectangle on the screen to represent the health the player has. The colour of the rectangle is decided with an r,g,b value of which the red is the value of the player's health. This way as the player loses health the bar gets progressively more red, this is a nice visual touch.. **Line 3** sticks the image surrounding the health bar on top. It makes it look nice.

```
1. def drawHealthBar():
2.
   pygame.draw.rect(win,(255,(playerHealth),0),(250,205,playerHeal
   th,60))#health bar will go more red as player gets lower health
3.    win.blit(healthbar,(200,200))
```

13.

*Testing and errors:**(inc. description of error encountered, fixing and justification of fix)***
Originally I didn't have **lines 6 and 7**

```
if playerHealth<0:
    playerHealth  = 0
```

so when I ran the code if the players health went below zero I got an error for an invalid colour argument. I realised this is because the player's health is passed into the rgb value for the drawHealthBar function, and you cannot have an rgb value which is negative. So I then put in **lines 6 and 7** to fix it.

Above is testing to make sure that the health bar doesn't go below zero, which it doesn't.

Test Number	Thing to be tested/ inputs	Expected outputs	Success?
1	Taking damage- use movement keys to collide with an enemy or lava block.	Health bar will decrease.	PASS
2	Health bars are capped at a maximum and minimum value. Walk into lava using movement keys and stay there.	Health value does not go below 0. Bar doesn't become drawn in the negative direction(as health is proportional to the width of the bar in pixels.)	PASS

User feedback:
My test user Joe said that he likes the way the health bar steadily changes colour. It adds a nice aesthetic touch. He also thinks that the health lost for each interaction is suitable, and that it provides a nice level of difficulty, whilst still making the user feel able to complete the levels.

Reflections:
I have realised with my damage blocks array, I can easily include spikes into my game as the functionality is all there, it wouldn't be that hard to implement. It could add nicely to the levels like a tunnel you need to go through, but spikes are blocking it so you have to get ammo and destroy the spikes first. This element of strategy could be good.
-have the damage blocks array so can include things like spikes as well, could be pretty cool, wouldn't be too hard to implement as all the functionality is there.

This took 2 attempts. One for the initial code and one to fix the problem of the invalid colour argument outlined above.
This is success criteria 16(lava)**completed**, and criteria 8&9(health bars)-**completed**

8	Give the player multiple lives or a health bar.	This ensures that they will not die immediately/ be able to play for a substantial amount of time.
9	Display lives/health bar.	This is very important for the player to know, as it will influence how they play the game. It would be very frustrating to not know how much health they have; therefore, a sort of health bar will be important to my game.

16	Have at least one damaging block. E.g. lava the player can fall into or spikes they can walk on.	An extra detail for the player to be aware of, making the game more interesting.

<u>Level objectives, coins and points system:</u>

Objective:

- To have coins and crystals in the level that I can collect. The coins will work as a means of collecting points and the crystals will be a level objective.
- Once all crystals are collected the level will be completed.

Code explanation:

Creating the collectable coins and crystals

Coins and crystals will be added to the game the same as any other tile, so I can just have them in my text file and they will be added automatically. However as I don't want projectiles to be blocked by these things I will add them to a cantCollideList, so i can check for collisions later with tiles not in this list.
I also want an animation for the coins so they appear to sparkle, I will do this the same way I have for walk animation. By having two images that are placed on top of the underlying rect in turn it creates the animation. **Lines 6-8** are from another code section that is responsible for drawing the level. I have included it here so it can be seen how the coincounter is responsible for changing the image on **line 6**. The purpose of the oneTimeLevelConstructor conditional loop is to make sure that I am not creating infinite coins. The crystals are created in a similar way but

they do not have an animation as I dont think this will be necessary and I do not have the time to design animations for everything.

```
if coinCounter>23:  #sets up the coin animation
            coinCounter = 0
coinCounter+=1


if tile == "m" : #creates the coin objects and draws their images onscreen
win.blit(coinImages[coinCounter//12-1],((x-1)*TILE_SIZEX, y*TILE_SIZEY))
     if oneTimeLevelConstructor: # only adds the coins once
        coins.append(pygame.Rect((x-1)*TILE_SIZEX, y*TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))
```

The code for collecting the coins is shown below. This is exactly the same for collecting the crystals.

```
for coin in coins:  #for every coin instance of the class
      if player_rect.colliderect(coin):  #if the player is touching the coin
               playerMoney += coinValue   #the player gets money
               coins.pop(coins.index(coin))  # the coin is deleted so it cant be collected again

game_map[(coin[1]//TILE_SIZEY)][(coin[0]//TILE_SIZEX +1)] = "
"#removes coin image from the game map so wont be redrawn even if already taken by player
```

Completing the level once all crystals have been collected

I now needed code to check if the player has collected all the crystals in the level and if so, to move to the next level. This is shown below. There is also the resetting of many values such as playerMoney, enemiesKilled ect, but I haven't shown this in order to save space.

```
if crystalsGathered ==  totalCrystals:# if level objectives completed
            currentLevel +=1#move to the next level
            game_map = levels[currentLevel-1]
```

Testing and errors:

I had to check that the coins disappeared when encountered, and that they gave the correct

amount of money. I also had to check that when all the crystals were collected the game moved onto the next level. This all worked out as planned and there were no edits needed.

The player collects a coin, and it then disappears.

Test Number	Things to be tested/ inputs	Expected outputs	Success?
1	When the player touches crystals they disappear.	When rect objects collide, the coin/ammocrate/crystal disappears.	PASS
2	The corresponding value increases	The counter which will be displayed at the top of the screen is incremented.	PASS
3	As well as disappearing they get deleted from level to stop them from being reused whilst invisible.	When the player moves over the spot the object once was in nothing happens(to the counters)	PASS
4	Level will only end when all crystals are collected	Collect crystals one by one and only when the last one is collected will the game move to the next level.	PASS

User Feedback:

My user said that he thinks it could be cool to be able to spend the coins on items, I dont have the skill to implement this feature in the time I have available, therefore it will just have to be a form of collecting points unfortunately.

Reflections:

Perhaps the crystals were not needed in order to complete the level, I could have just made it so you had to kill all the enemies, due to time constraints this would have been easier. However interviewees said that they liked the idea of something that you had to collect and now I have

this so that is a good thing.
This code took me 1 attempt, as no errors were detected.

This is success criteria 13 & 11 (for coins) completed, and 14(crystals/level objectives)completed

14	Have at least three crystals for the player to collect in order to pass the level.	This will mean that levels will not be too easily beaten, making for a more challenging experience

13	A way to score points by for example defeating enemies (cake soldiers), or collecting coins.	Will encourage players to go for the challenge of defeating enemies, in order to achieve a higher score.

11	Must have collectable coins in every level.	This can be used in calculating the high score, and maybe also time spent on each level. Adds another factor to the game as well as possibly being used to buy weapons/ upgrades. This will make for more interesting levels, something Joe said was important in his interview.

Attacking, crouching:

Objective:

- To be able to attack
- To be able to crouch to get through one-block high gaps.

Code explanation:
Being able to attack
Lines 1-8 determine if the player is attacking(if the enter key is being held) and if so depending on which way the player is facing will change the dimensions of a rect object(which represents the sword) to be placed in front of the player. If the player isn't attacking then the sword is moved off the screen so it doesn't affect anything, this is in **line 9-11**. **Line 12** then goes through all the instances of the Enemy class and checks to see if the sword rect is touching any of them and the corresponding counter is equal to zero(if not damage is dealt too fast, the explanation of similar counters and how they work is in the code explanation of the enemies milestone). If both

of these booleans are true then the enemy should take damage, and I have also made them take a small amount of knockback on **line 15**. I also have code to make the enemies take damage due to things like lava, this is **line 16-18**. **Line 19 and 20** will kill the enemy if their health reaches zero.

```
1.        if isPlayerAttacking:#if the player is attacking
2.            if lastFacing == 1:#draws sword in correct position
   relative to body
3.                sword[0] = player_rect[0]+ 64
4.            else:#draws sword in correct position relative to body
5.                sword[0] = player_rect[0]-(32+sword[3])
6.            sword[1] = player_rect[1]+ 32
7.
8.            pygame.draw.rect(win,red,sword)#draws object onscreen
9.        if not isPlayerAttacking:#if not attacking then moves the
   sword offscreen so as not to accidentally damage enemies
10.             sword[0] = -100
11.             sword[1] = -100
12.       for enemy in enemies:#if the sword touches an enemy
13.             if sword.colliderect(enemy) and
   playerDealDamageCounter == 0:
14.                 enemy.health -= 50#damage enemy and deal
   knockback
15.                 enemy.rect[0]+=10*lastFacing
16.             for block in damageBlocks:#if enemy touches e.g.
   lava then damage them
17.                 if block.colliderect(enemy):
18.                     enemy.health -=10
19.             if enemy.health <=0:# if enemy dies get rid of the
   object
20.                 enemies.pop(enemies.index(enemy))
21.         playerDealDamageCounter +=1
22.         if playerDealDamageCounter >= 10:# rests the damage
   counter
23.             playerDealDamageCounter = 0
```

Being able to crouch

The code below is for my crouching. This code is very similar to the walking code which is in the moving/jumping milestone with only a few differences. It has a different animation and it also changes the height of the rect object representing the player on **line 2**. This is in order to allow the player to pass through one-block high gaps if crouching, as desired in my objective.

```
1. if crouching:
2.    player_rect[3]=50# if crouching change the players height
3.    if lastFacing == 1:
4.        playerImage = crouchRight1
```

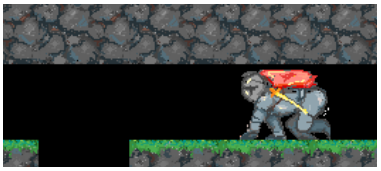
```
    else:
        playerImage = crouchLeft1
    if crouchCycle>23: # updates the crouch animation
        crouchCycle = 0
    crouchCycle +=1

    if moving_right:# move the player and display correct animation
        player_movement[0] = 2
        playerImage = crouchingRight[(crouchCycle)//12-1]
        lastFacing = 1
    if moving_left: # moves the player left and displays the correct animation
        player_movement[0] = -2
        playerImage = crouchingLeft[(crouchCycle)//12-1]
        lastFacing = -1

```

*Testing and errors:**(inc. description of error encountered, fixing and justification of fix)***
In testing my code I noticed that if I let go of the crouch button and the player was still in a one block high gap, they would stand up despite not being able to. This resulted in a collision between the player and the floor being detected, and my code automatically tries to rectify this by moving the player to the top of this block. I do not want this and it is very similar to the teleportation problem I mentioned in my ladders/climbing milestone. To fix this I needed a way to check if the player should be able to stand up, and if not, to not allow them to. Therefore I wrote the code seen below.

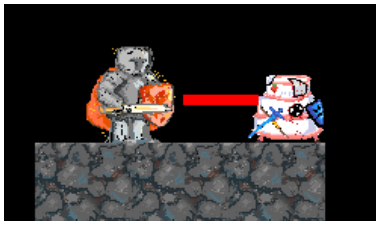
```
def canStand():
    global isCanStand
    isCanStand = True# holds boolean value representing if player can go off crouch or not
    hitList = []
    player_rect[3] = 64                #sets player to non crouch position
    player_rect[1] = player_rect[1]-14 #
    playerTop = player_rect[1]
    playerBottom = player_rect[1]+ player_rect[3]
    tileBottom = 0# initialises the variable
    for tile in tile_rects:
        if player_rect.colliderect(tile):
            hitList.append(tile)

    for tile in hitList:                #if the player is touching any tile
        tileBottom = tile[1]+tile[3]
```

```
16.          if playerTop<=tileBottom:        # if the players head
   is above the bottom of the tile
17.              if playerBottom>tileBottom:# if the players bottom
   is below the bottom of the tile
18.                  isCanStand = False    #then the player cant
   stand up
19.
20.                  player_rect[3]=54
21.                  player_rect[1]= player_rect[1]+14
22.      return isCanStand
23.
```

It works by checking to see if the player was standing up, if there would be a collision between the player and any block, and if so, it compares the y-level of the players head to the bottom of the block. If the players head is above the bottom of the tile and below the top(i.e inside the block) then the boolean isCanStand is set to False, and the player isn't allowed to stand up even if they release their finger from the crouching key.



Despite the crouching key not being held in this image, the player now no longer stands up as they should not be able to, thanks to the addition of the code above.



In this testing, I am looking at the knockback and the damage rate applied to the enemies by the sword(the red rectangle, I will give it a sprite later to make it look nicer.) I have found that after a little tweaking of the damage counters they take damage at a good rate.

Test Number	Things to be tested/ inputs	Expected outputs	Success
1	Dealing damage- use the sword keybind and walk into an enemy with it.	Enemy health will decrease, they will eventually change images to their corresponding health percentages, and eventually die(disappear from screen). Enemy kill count should increment if this happens.	PASS
2	Applying appropriate amounts of damage.	For a collision with an enemy, the taking damage counter should stop the player from taking too much damage. Health bar should only go down around a tenth(this value is	PASS

		subject to change as I see fit). Enemy should also not die immediately.	
3	Player can only un-crouch at appropriate times	When crouch key is released if under one block high gaps the player wont uncrouch	PASS

User feedback:

The test user Joe said that he would like there to be a sword animation, which I currently don't have, and maybe even a sword cooldown so you cant use it constantly. These are things that I can definitely implement, and I will have some form of animation for the sword by the end of my project so that it is not just a red rectangle.

I have done this in the polishing features milestone but am including a screenshot of it here for reference of my development based on user feedback.

Reflections:

If the player releases the crouching key whilst still in a tunnel, they are not allowed to stand up, this means I could either make it so they stand up automatically when they leave the tunnel(more work but allows for smoother gameplay) or press the crouch key again when they are out of the tunnel to stand up(much easier to implement, but less smooth gameplay) Due to time constraints I have chosen the second option, as I will be able to implement it easily and it will not take much testing/ error fixing to make it work, and I am limited on time.

This milestone took 2 main attempts to complete. The first for the initial algorithms, and the second to provide the addition of the function to detect whether the player should be allowed to stand up.

This is part of success criteria 1, the crouching(“s”) is **completed**, attacking, success criteria 7 **completed**, but can be improved upon,(and is in the polishing features milestone)

1	Must have WASD for movement, with the addition of spacebar for jump(as w will be for ladders.)	Many people from the target audience have expressed their wish for these keys to control the movement of the player, due to their common use in pc games

7	A melee attack the player can use at close range. Time permitting, a cooldown on this weapon.	A basic attack for before better weapons is discovered or bought between levels, to make sure that early levels are playable.

Importing level details from text files:

Objective:

- To be able to import levels from a text file, instead of having to store all of the details in arrays in my program code which clutters it alot.

Code explanation:

Being able to import level details from a text file

Lines 1-6 allow me to later in my code when loading another level be able to find the name of the text file corresponding to a certain level number so I can then load it. **Lines 9-13** set up 1 dimensional arrays for all the levels, and **lines 14-23** turn them into 2 dimensional arrays to represent the screen with each element being a space where a tile can be drawn.

```
1. levelsAndNames=[ # sets up the array to match the names of the
2. [1,"level1.txt"],#files to the corresponding levels
3. [2,"level2.txt"],
4. [3,"level3.txt"],
5. [4,"level4.txt"],
6. [5,"level5.txt"]
7. ]
8.
9. level1 = [" "]*20# initialises the level arrays for all the
10.   level2 = [" "]*20# details to be imported into
11.   level3 = [" "]*20
12.   level4 = [" "]*20
13.   level5 = [" "]*20
14.   for i in range(20):
15.      level1[i] = [" "]*32
16.   for i in range(20):                    #sets up all the level
   arrays
```

```
17.      level2[i] = [" "]*32
18.  for i in range(20):      # finishing set up of all the level
   arrays
19.      level3[i] = [" "]*32
20.  for i in range(20):
21.      level4[i] = [" "]*32
22.  for i in range(20):
23.      level5[i] = [" "]*32
24.  levels = [level1,level2,level3,level4,level5]
```

I then need a function to import the levels from my text files into these arrays that I have now created, this is shown below. **Lines 2-5** take the level number requested, find its location in the levelsAndNames array created above, and find the name of the corresponding text file which holds the details of that level. **Line 6** then opens this text file and **lines 7-12** are responsible for separating these text files by all the commas in it(these commas separate each tile which is represented by a letter) and then insert each of these letters into the arrays created by my code above.

```
1. def importLevel(levelNumber):
2.    levelToLoad = ""
3.    for i in range(len(levelsAndNames)):#determines the level
   that should be loaded and the text file that relates to this.
4.        if levelsAndNames[i][0] == levelNumber:
5.            levelToLoad = levelsAndNames[i][1]
6.    file = open(levelToLoad,"r")
7.    x = 0
8.    for line in file:
9.        content = line.split(",")#seperates the file at all the
   data
10.         for i in range(len(content)-1):
11.             (levels[levelNumber-1])[x][i] = content[i]#extract
   the level data and place in the array
12.         x +=1
13.     return(levels[levelNumber-1])
```

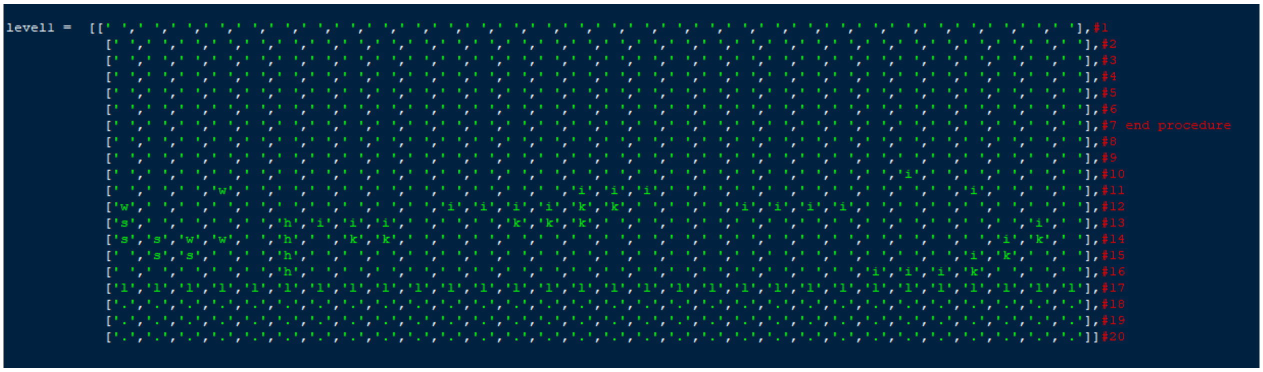
Testing and errors:***(inc. description of error encountered, fixing and justification of fix)***

I realised that if my code was in separate folders to the program then my importLevel function would no longer work as it wouldn't be able to locate the text files. I then put these levels into a separate folder and changed the code on **line 6** to :

```
file = open(("leveldetails/" + levelToLoad),"r")
```

As this will locate the correct folder and then the level text file within it. After testing this code with everything in its respective folder it worked fine.

Now the levels have gone from being stored in the code in an array like this:



To being in a text file like this:

level3 - Notepad

File Edit Format View Help

```
k, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,k,
k, , , , , , , ,h, , , , , , , , , , , , , , , , , , , , , , ,k,
k, , , , , , , ,h, , , , , , , , , , , , , , , , , ,c, , , , ,k,
k, , , , , , , ,h, , ,c, , , , , , , , , , , , , , ,i,i, , , ,k,
k, , , , , ,i,i,h,i,i,i,i, , , , , , , ,e, ,c, , , , , , , , ,k,
k, , , , , , , ,h, , , , , , , , ,i,i,i,i,i,i,i, , , , , , , ,k,
k, , , , , , , ,h, ,k, , , ,k, , , , ,k,k,k,k, , , , , , , , ,k,
k, , , , , , , ,h, , ,k, ,k, , , , , , ,k, , , , , , , , , , ,k,
k, , , , , , , ,h, , , ,k, , , , , , , , ,e, , , ,e, ,c, , , ,k,
k, , , , , , , ,h, , , , , , , , , ,k,k,k,k,k,k,k,k, ,i, , , ,k,
k, , , , ,k, , ,h, , , , , , , , , , , , , , , , , , , , , , ,k,
k, , , , , , , ,h, , , , , , , , , , , , , , , , , , , , , , ,k,
k,i,i,i,i, , , ,h, , , , , , , , ,e, , , , , , , , , , , , , ,k,
k, , ,k,k,k, , ,h, , , , , , , ,k,k,k,k,k,k,k,k,k,k, , , , , ,k,
k, , , , , , , ,h, , , , , , , , , , , , , , ,e, , , , , , , ,k,
k, , , , , , , ,h, , , , , , , ,i,i,i,i,i,i,i,i,i,i,i,i,i,i, ,k,
k,l,l,l,l,l,l,l,l,l,l,l,l,l,l,l,l,l,l,k,k,k,k,k,k,k,k,l,l,l,l,k,
k,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,k,
k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,.,.,.,.,.,k,k,k,k,k,k,k,k,k,k,k,
k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,k,
```

Test Number	Things to be tested/ inputs	Expected outputs	Success?
1	The correct level loads	The levels are loaded and played in the intended order	PASS

User feedback:

There is none for this milestone apart from less cluttered folders, as this does not change the game in itself in any way, only the code, and so the user has little experience with the changes from this milestone.

Reflections:

This is much more maintainable now if I want to add more levels in the future. The extra folder is nice as it stops my main one from becoming too cluttered, Charlotte says that she appreciates this also so it should be a big success with my stakeholders.

This code took 1 attempt to complete

This helps with the completion of success criteria 2, as it allows for me to easily design and store levels. I have now made 5 different levels **completed,** and each is a different difficulty(criteria 3)-**completed**, with platforms at different heights(criteria 17)**completed**

17	Have many different platforms separated from each other and at different heights.	Forces player to make use of the jump ability or ladders, adds the extra threat of falling into lava.

3	Levels increase in difficulty, with one very hard level at the end.	As the player progresses Joe has said that it will be important that the game remains challenging, and so I have decided incrementing difficulties will be necessary in the level.

2	At least 5 different levels.	In order to keep the game interesting Joe has stated in his interview that it should have multiple levels, preferably 5 or more.

Ranged weapons

Objective:

- To be able to have a ranged weapon that the player can use.
- Also to have some sort of ammo count to force the player to also use the sword feature, otherwise the levels would be too easy if you can always attack from range.

Code explanation:

Creating the ranged weapon

I need a new bomb/projectile class so I can create as many instances as I like. Below is the constructor of my class and it is responsible for calculating the movement vectors for my projectile. The purpose of the collisionEnemy and collisionTile attributes is so I can assign these the details of a tile/enemy that it hits. Then later on I can check these attributes and if they are no longer equal to “ ” then I can delete the corresponding tile or damage the corresponding enemy.

```
1. class Bomb():
```

```
2.    def
   __init__(self,length,player_rect,damage,velx,vely,image,gravity
   =0.1,collisionTile = " ",collisionEnemy=" "):
3.        self.damage = damage#intialise all the class atributes
4.        self.collisionEnemy = collisionEnemy
5.        self.collisionTile = collisionTile
6.        self.length = length
7.        self.image = image # sets up the image attribute
8.        self.collided = False# sets up all the other attributes
9.        self.collidedEnemy = False
10.       self.collidedTile = False
11.       self.gravity = gravity
12.       self.x = player_rect[0] + (player_rect[2]//2)#sets
   starting co-ords of projectile to center of player
13.       self.y = player_rect[1] + (player_rect[3]//2)#
14.
15.       self.velx = velx//25#divide by 25 otherwise too big
16.       self.vely = vely//25
17.
18.       if abs(self.velx)<1:          #
19.           if self.velx>=0:          #
20.               self.velx = 1         #stops the velocity from
   being too small
21.           else:                     #
22.               self.velx = -1        #
23.       if abs(self.vely)<1:          #
24.           if self.vely>=0:          #
25.               self.vely = 1         #
26.           else:                     #
27.               self.vely = -1        #
```

Below are the draw and and move functions. The draw function works like every other draw function of mine, it draws the underlying rect that I use for collision detection and then sticks the corresponding sprite image on top.
The move function simply adds the corresponding x and y vectors to its current position to allow the projectile to move. It also adds acceleration due to gravity to the y vector, but this is capped at a certain value as it is in real life due to air resistance(I want the game to feel realistic with the projectiles.)

```
1.    def draw(self):# the draw method for the projectile
2.        self.rect =
   pygame.draw.rect(win,red,(self.x,self.y,self.length,self.length
   ),1)
3.        win.blit(self.image,(self.x,self.y))
4.    def move(self):#moves the projectile
```

```
5.        self.x+= round(self.velx)#round function necessary as
   cant move by half a pixel ect.
6.        self.y += round(self.vely)
7.
8.        self.vely += self.gravity # the gravity on the projectile
9.        if self.vely>5:
10.             self.vely = 5
```

Below is the collisionCheck function. The purpose of this is to check if the bomb collides with a tile or enemy, and if so it will return the details of what it collided with and an identifier for it so it can be damaged if its an enemy or deleted if its a block.

```
1.    def collisionCheck(self,tiles):#checks for collisions with
   any blocks
2.        for tile in tiles:#goes through all tiles and checks for
   collisions
3.            if self.rect.colliderect(tile[0]):
4.                self.collidedTile = True
5.                self.collisionTile = tile
6.        for enemy in enemies:#checks for collisions with any
   enemies
7.            if self.rect.colliderect(enemy):# checks for
   collisions with enemies
8.                self.collidedEnemy = True
9.                self.collisionEnemy = enemy
10.         return
   (self.collidedTile,self.collidedEnemy,self.collisionTile,self.c
   ollisionEnemy)
```

Creating the ammo limiting feature

On **Line 1** I check to see if the player should be able to fire, this is if the counter(this prevents them from firing too quickly) is zero and there arent too many bombs on screen and the player actually has ammo, then they are allowed to fire.

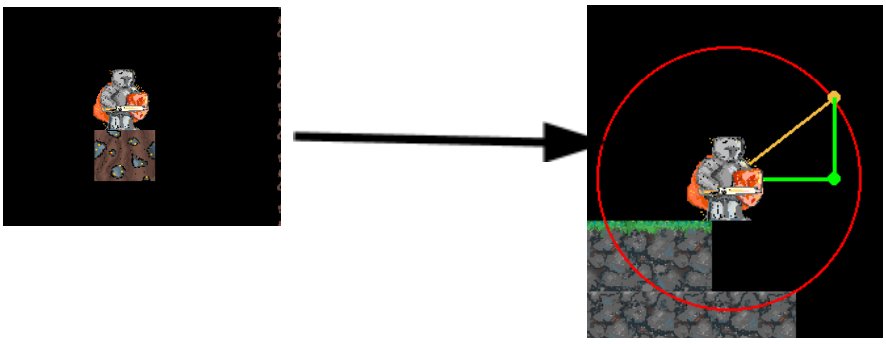
A projectile is fired(a member of the bomb class is instanciated) and the counter is reset for the next shot.

```
1. if  len(bombs)<4 and shotsFiredCounter == 0 and playerAmmo>0
   and aimed :    # if the player should be able to fire
2.
   bombs.append(Bomb(15,player_rect,299,aimVectorDeltax,aimVectorD
   eltay,bombImage))      #fire
3.    playerAmmo-=1   # decrease ammo
4.    shotsFiredCounter += 1
5.
6. shotsFiredCounter+=1         # handle the firing counter
7. if shotsFiredCounter >7:
8.   shotsFiredCounter = 0
```

*Testing and errors:**(inc. description of error encountered, fixing and justification of fix)***
It took some trial and error to finally settle on the limit on the y speed for the projectile(essentially its terminal velocity on the y-axis of the screen). I went through values where the projectile fell too fast. This made it impossible to aim, and when it fell too slowly, the range was too great and aiming was too easy.
When testing the aim, I noticed it wasn't very intuitive. It was hard to tell where you were aiming as the projectile path is an arc and therefore the projectile wont naturally go through the pointer(I could have made it so it would, but this would be too easy for the player and therefore remove an element of challenge that my stakeholders would enjoy). To make the aiming easier I implemented the beneath code to show the initial path of the projectile, with a circle of radius relating to the speed of the projectile.

```
1. aimLineEndPoint = [(aimCircleCenter[0] +
   aimDeltax*(radiusAimCircle/aimLineLength)),(aimCircleCenter[1]
   + aimDeltay*(radiusAimCircle/aimLineLength))]#initialising all
   of the variables to be used in drawing the aim lines
2. pygame.draw.circle(win,red,(aimCircleCenter[0],aimCircleCenter[
   1]),radiusAimCircle,2)     #draws the red circile
   pygame.draw.line(win,gold,(aimCircleCenter[0],aimCircleCenter[1
   ]),(aimLineEndPoint[0],aimLineEndPoint[1]),3)
3. pygame.draw.circle(win,gold,(aimLineEndPoint[0],aimLineEndPoint
   [1]),5)
4. pygame.draw.circle(win,green,(aimLineEndPoint[0],aimCircleCente
   r[1]),5)
5. pygame.draw.line(win,green,(aimCircleCenter[0],aimCircleCenter[
   1]),(aimLineEndPoint[0],aimCircleCenter[1]),3)#draws green lines
6. pygame.draw.line(win,green,(aimLineEndPoint[0],aimCircleCenter[
   1]),(aimLineEndPoint[0],aimLineEndPoint[1]),3)
```

What this code does is draw the following lines on the screen:



The small yellow ball is where the mouse is, the radius of the red circle is proportional to the total velocity, and the green lines to the horizontal and vertical velocities respectively.

Test Number	Things to be tested/ inputs	Expected outputs	Success?
1	You cannot use more ammo than you have. Inputs: mouse buttons.	You cannot fire a projectile when the ammo counter is 0. Ammo counter will not go below zero.	PASS
2	The path the projectiles follow	The players projectile follow a realistic path	PASS
3	Destroying blocks/damaging enemies	Destroys the blocks or damages the enemies depending on what it hits	PASS

User feedback:

They liked the addition of the above circles and lines, saying that it was very difficult to aim otherwise, but the circles make the controls more intuitive and easy to pick up, whilst still creating a good enough difficulty level, as the lines only show the initial path of the projectiles before they have been affected by gravity, and so aiming is not too easy.

Reflections:

I can make the aim “path plotters” look nicer by doing designs for the circles and lines, however as I am time pressured, this will only be implemented if I have time after completing the more important functionality first, the basics need to be in place before I can start polishing everything.

This milestone took 2 attempts. 1 for the initial code, and another to include the aim lines and circles as requested by my user.

This is success criteria 6, ranged attack **completed**, and criteria 5(powerup[the ranged weapon will function as a powerup as ammo is limited])**completed**

5	Power ups: at least one.	In their interview Joe and Charlotte stated that power ups should be integrated into the gameplay, in order to add new variables and play-styles to make it so there are many ways to beat a level, so they can be played multiple times without getting old. This is very important to my game as a problem with many platformers is they become boring after a few hours, as stated by many of my stakeholders.

6	A ranged attack that the player can use.	If the character has a ranged weapon, this will make for more interesting levels, and therefore contribute to solving my problem of keeping the player entertained.

Highscores:

Objective:

- To have a highscore option on the menu that will display the top 5 scores. To do this I need code to save scores to a text file when the player dies or completes all the levels. I then need a function to extract and then order the scores in a list. Finally I need to display the top five on the high score screen.

Code explanation:

Calculating and displaying the top 5 scores

The following code will open the highscore file located in the correct folder. It will then create a new line(this allows me to separate the scores easily) and it will write the recent score in with a comma after it. The comma is necessary as it allows me to split the data on this line at the "," so I can get rid of the invisible new line character. This code will be utilised when the players health is <= 0 or they have completed all levels.

```
1. file = open("highscores/highscores.txt","a")#opens highscores
   for writing in
2. file.write("\n")
3. file.write(str(totalScore)+",")#writes the scores to the file
4. file.close()#closes the file
```

The following code will extract the scores, then sort them in descending order.

```
1. file = open("highscores/highscores.txt","r")#opens file
2. lines = file.readlines()# extracts data to variable lines
3. for i in range(len(lines)):
4.         details = lines[i].split(",")
5.          scores.append(int(details[0]))
6. scores.sort(reverse = True)#extracts scores and sorts them in
   descending order
```

The following code will take the top 5 elements of the scores array, and for each one assign it a place(e.g. The top score) and decimal places to take up screen width and then the final score. Each of these will be added to a new array called topFiveScores.

```
1. for i in range(5):
2.     text = str(i+1)
3.     points = abs((screenWidth/fontSize) -
   (len(str(scores[i]))+1)) #the number of dots to put between the
   place and the score
4.     for j in range(int(points)):
5.         text = text + "."
6.     text = text + str(scores[i]) # this and few lines above make
   the score thingy e.g. 1.................450
7.     topFiveScores.append(text)
8.     text = ""
9. print(topFiveScores)
10.   oneTimeScoreCalculation = False
```

The following code will draw the highscores on the screen.

```
1. for i in range(5):# for loop to display top 5 scores
2.
   draw_text(font1,topFiveScores[i],black,fontSize,screenWidth//2,
   200+125*i)#displays the top 5 scores
3. pygame.display.update()
```

*Testing and errors:***(inc. description of error encountered, fixing and justification of fix)**

There was a problem in that whilst inside of the highscore loop, i noticed that the same top high score was displayed 5 times, despite there only being one of them and no duplicate scores. I realised that this was because the code kept adding all the scores to the score array over and over, and then sorting it so the top 5 spaces were composed of only the top one.

To fix this I created a variable called oneTimeScoreCalculation, and only took scores out of the file and put them into the array if this was True. I then set it to False when done. This is shown below

```
        if oneTimeScoreCalculation:
#extracts scores from file

            file = open("highscores/highscores.txt","r")
            lines = file.readlines()
            for i in range(len(lines)):
                details = lines[i].split(",")
                scores.append(int(details[0]))
            scores.sort(reverse = True)
```

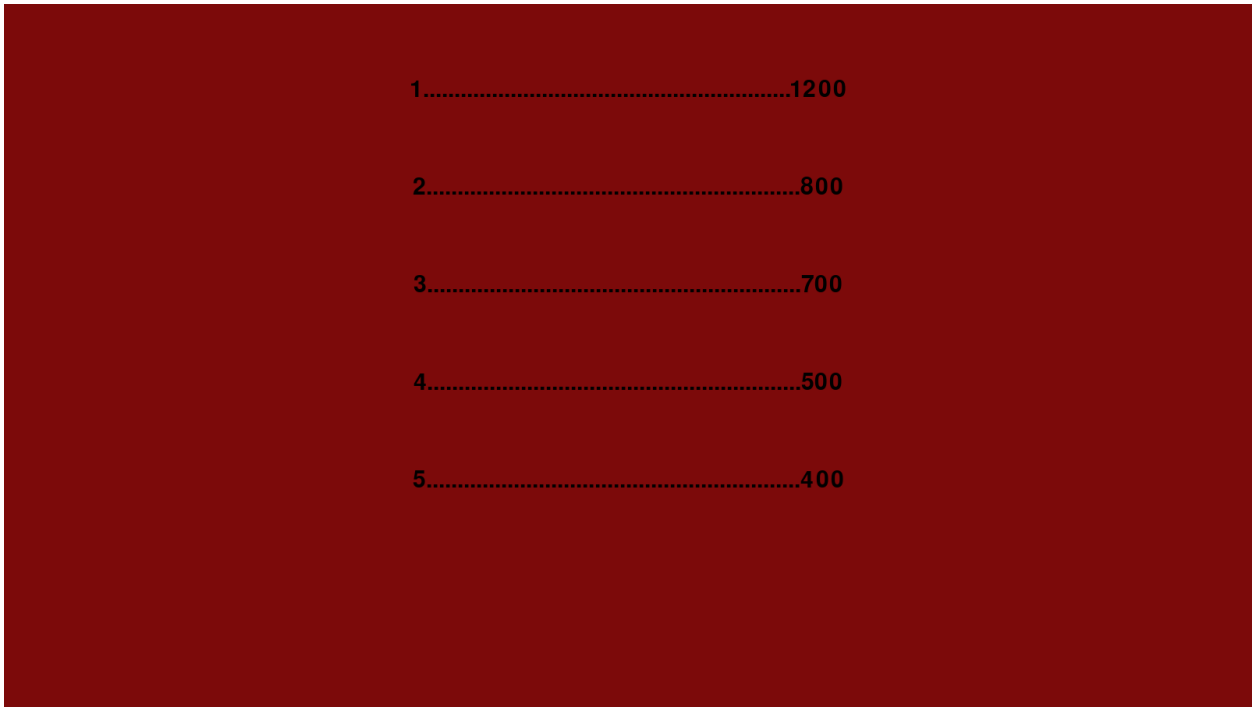
With further testing to see if this fixed the problem it became apparent that as this allows the highscores to be calculated only once, if you try re-checking the scores after getting a new high score it will not appear. To fix this I simply reset onTimeScoreCalculation to True when the highscore menu is selected.

Test Number	Things to be tested/ inputs	Expected outputs	Success?
1	Highscores are displayed	Top 5 scores shown on screen	PASS
2	Displayed in correct order	The scores are sorted numerically, largest first.	PASS

User feedback:

My stakeholders think that the design of the high score screen could use some improvement as

at the moment it looks like this:



Reflections:
Time permitting I will add a nice background and perhaps trophy drawings by the number one spot to be more engaging to my target audience and encourage them to play more to hold the high score.
This code took 2 attempts.
This is success criteria, number 10, highscores **completed**

10	A menu with highscores, level select and play option.	Player should be able to see their high scores, or choose which level they want to play based off of their difficulties.

Sound effects:

Objectives:

- To have sound effects play for the following reasons: taking damage, tile exploding, collecting coins, jumping, attacking, successfully blocking a ranged attack with the shield.

This will make the game more immersive for my target audience.

Code explanation:

Loading and playing the sound effects

The following code will load all the sound effect files (These sound effects have been taken from the following website: https://sfxr.me/), and then save them each to the corresponding variable name.

```
1. pygame.mixer.music.load("soundfx/damageSound.wav")#loads the
   sound effect
2. damageSound =
   pygame.mixer.Sound("soundfx/damageSound.wav")#saves it to the
   damageSound variable

   #this is the same process for all of the following sounds

3. pygame.mixer.music.load("soundfx/explosionSound.wav")
4. explosionSound =
   pygame.mixer.Sound("soundfx/explosionSound.wav")
5. pygame.mixer.music.load("soundfx/coinSound.wav")
6. coinSound = pygame.mixer.Sound("soundfx/coinSound.wav")
7. pygame.mixer.music.load("soundfx/jumpSound.wav")
8. jumpSound = pygame.mixer.Sound("soundfx/jumpSound.wav")
9. pygame.mixer.music.load("soundfx/attackSound.wav")
10.   attackSound = pygame.mixer.Sound("soundfx/attackSound.wav")
11.   pygame.mixer.music.load("soundfx/shieldSound.wav")
12.   shieldSound = pygame.mixer.Sound("soundfx/shieldSound.wav")
```

The following code is an example of how I play the sound effect when I need it.

```
1. if isPlayerAttacking:#if attacking
2.   attackSound.play()# play attack sound
```

Testing and errors:

This milestone was implemented very successfully and quickly due to the simplicity of the code. All I had to do was look up how to use the music module on the pygame frontpage (https://www.pygame.org/docs/ref/music.html) so I could implement my code. There were no errors and when testing all the sound effects played when I wanted them to.

Test Number	Things to be tested/ inputs	Expected outputs	Success?
1	Sound effects play	Sound effects at a suitable volume	PASS
2	Correct sound effects play	The right sound effect for the scenario is played	PASS

User feedback:
The users are very happy with the sound effects, they say that it reminds them of the games they used to play when they were younger, which is exactly what I intended. When I asked them they agreed that it made the experience more immersive and enjoyable in general.

Reflections:
I was slightly concerned that there were too many sound effects and it would be a bit hectic, but my stakeholders (who are an accurate representation of my target audience) think that it is fine. Still if I find others would prefer them to be taken out, I can remove them or even time permitting create an option to toggle music on and off.
This code took 1 iteration.
This is success criteria 12. Sound effects **completed**, background music **abandoned** as background tracks will mean too much noise and will make the game hectic.

12	Sound effects and background music. At least 2 different music tracks to reflect difficulty.	Increases the level of immersion making for a more fun session.

Instruction screen:

Objective:
- To have an option in the menu that when clicked will allow you to read instructions on the game.

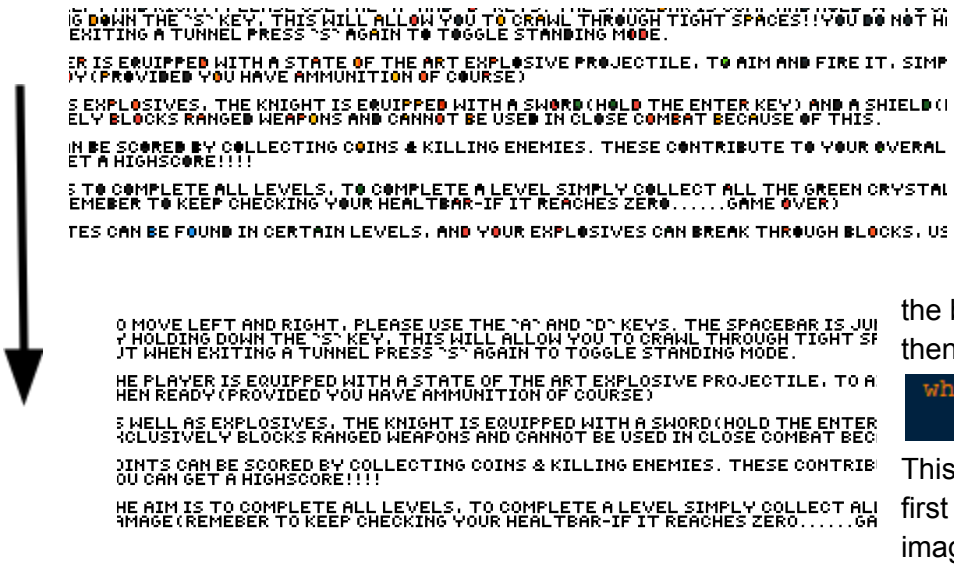
Code explanation:
Creating the instruction screen
To do this I created an instructions loop inside of the main loop (but outside of the game loop where all the other menu options like highscores are), That will run if the Boolean instructions is set to true. This happens when in the menu loop this option is selected(It was very easy to do this as all the functionality was there from the other menu options, I only needed to add a line or two of code and so am not displaying that here.) I created an image with the instructions written out on them. This image is displayed on **line 3**, and the rest of the lines are just adding images to take up some of the blank space on the screen. **Lines 15 to 21** are in every menu option loop and simply allow you to return to the main menu.

```
1.    while instructions:#if on instructions screen
2.        win.fill(white)#add all of the images
3.        win.blit(instructionBackgroundImage,(0,0))
4.        draw_text(font1,"Press p to return to main
   menu",black,40,screenWidth//2,800)

   # beneath code draws all of the images

5.        win.blit(crystalImage,(700,600))
6.        win.blit(knightRight,(800,600))
```

```
7.          win.blit(coin1,(910,620))
8.          win.blit(crouchingLeft[0],(1000,620))
9.          win.blit(ladderImage,(1100,600))
10.         win.blit(ladderImage,(1100,654))
11.         win.blit(ladderImage,(1100,708))
12.         win.blit(ammoCrateImage,(800,700))
13.         win.blit(cakeAttackerImages[0],(945,700))
14.
15.         pygame.display.update()# updates the display to show the
    instructions and all of the images
16.         for event in pygame.event.get():#allow you to exit the
    instruction screen
17.             if event.type == KEYDOWN:
18.                 if event.key == K_p:
19.                     instructions = False
20.                     menu = True
21.         pygame.display.update()
```

*Testing and errors:**(inc. description of error encountered, fixing and justification of fix)***

When I first ran this code without **line 2**(which first covers the screen in the colour white) I was given the following image.



(I have cropped it). The colours in the middle of the text is from the main menu screen which can be seen through my instruction image through the blank pixels. To fix this I then added **the following code**

```
while instructions:
    win.fill(white)
```

This covers the screen in white first then adds the instruction image.

The problem is now fixed

Test Number	Things to be tested/ inputs	Expected outputs	Success?
1	Instructions are shown	When instruction option is selected they are shown	PASS

2	You can leave the instruction screen	When the exit key is pressed player returns to the main menu	PASS

User feedback:
One of my stakeholders said that the text was possibly too small to read easily and that there was a lot of text. The other thought that it was fine. However the text is necessary to explain the game objectives and also how to play, therefore it is necessary

Reflections:
As mentioned above in user feedback, this text is necessary and so there isn't anything I can do about it, however the users do have the option to not read the instructions if they want, so it is up to the user which is good as it allows them to make their own choice based on their own preferences.
This code took 1 main attempt, to write the algorithms and then test and fix the error of the background outlined above.
This is an addition to success criteria number 10, that I decided to implement to explain how to play the game. - addition **completed**

10	A menu with highscores, level select and play option.	Player should be able to see their high scores, or choose which level they want to play based off of their difficulties.

Polishing features, sword animation, shield animation

Objective:

- For the animations to look cleaner to make the game more visually appealing. To do this I will need around 3 different images to show the player attacking with a sword, and cycle through these every time the player attacks. For the shield animation, the player to be able to hold a shield and walk with it.

Code Explanation:

Creating the attacking animation.

The animations are coded very similarly to the walking and crouching ones ect. I load the images into an array, and when necessary I cycle through these. **Lines 1-4 and 6-9** will load the images in. **Lines 5 and 10** place these into an array.

```
# all this code does is load images and save them to the corresponding variables, then making 2 arrays for use in the animations
1.  attackImage1R = pygame.image.load("Images/attackImage1Right.png").convert_alpha()
2.  attackImage2R = pygame.image.load("Images/attackImage1Right.png").convert_alpha()
3.  attackImage3R = pygame.image.load("Images/attackImage1Right.png").convert_alpha()
4.  attackImage4R = pygame.image.load("Images/attackImage1Right.png").convert_alpha()
5.  attackAnimationRight = [attackImage1R,attackImage2R,attackImage3R,attackImage4R]
6.  attackImage1L = pygame.image.load("Images/attackImage1Left.png").convert_alpha()
7.  attackImage2L = pygame.image.load("Images/attackImage1Left.png").convert_alpha()
8.  attackImage3L = pygame.image.load("Images/attackImage1Left.png").convert_alpha()
9.  attackImage4L = pygame.image.load("Images/attackImage1Left.png").convert_alpha()
10. attackAnimationLeft = [attackImage1L,attackImage2L,attackImage3L,attackImage4L]
```

The following code identifies if the player is attacking, and if so depending on how long they have been attacking for(the attackCycle variable holds this) displays the corresponding image in the animation. When the player has been attacking for a certain amount of time(when attack cycle = 24 on **line 10**), the boolean on **line 1** is set to False, and the attack is finished. (It is an adapted version of the code in the “Attacking, crouching” section on page 17)

```
1. if isPlayerAttacking:#if player is attacking
2.     if attackCycle>23:#handles the animation cycle
3.         attackCycle = 0
4.     attackCycle+=1
5.     if lastFacing == 1:#displays the corresponding images in the animation cycle
6.         playerImage = attackAnimationRight[(attackCycle)//6-1]
7.     else :
8.         playerImage = attackAnimationLeft[(attackCycle)//6-1]
9.     attackSound.play()#play attack sound effect
10.       if attackCycle == 24:
11.           isPlayerAttacking = False
```

*Testing and errors:**(inc. description of error encountered, fixing and justification of fix)***

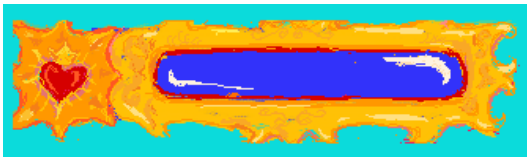
There are no test tables for this section from the “Identifying and justifying test data” segment of the design section. This is because this milestone was not a part of my initial design.

I realised that the attack feature was unfair in testing, in that you could just constantly attack and therefore the enemies presented no challenge. To fix this I needed to implement a weapon cooldown so it could not be used constantly. This involves displaying the cooldown(so the player knows when they can attack), and adding extra code so the attack algorithm is only followed if the cooldown has been completed. The code for this is below: The boolean on **line 3** is used to initiate the attack sequence as usual

```
1. if event.key == K_RETURN and weaponCooldown == 250:#if the
   player is trying to attack and the cooldown is completed
2.     weaponCooldown = 0
3.     isPlayerAttacking = True#let the player attack
```

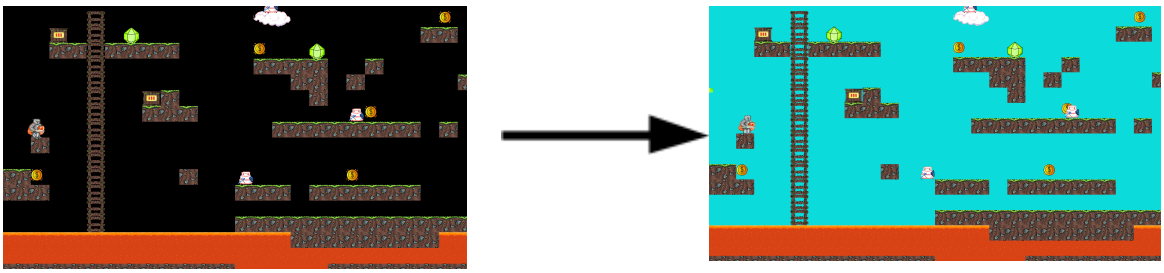
```
1. def drawWeaponCooldown(weaponCooldown):#draws the cooldown bar
2.     colour = (50,50,(weaponCooldown))
3.     pygame.draw.rect(win,colour,(450,5,weaponCooldown,60))
4.     win.blit(healthbar,(400,0))
```

Now the player can only attack when the cooldown is completed, the bar to display this also works nicely



User feedback:

My stakeholders said that the bar looked nice, however they noticed that the cooldown bar has a heart icon on it. This is because in order to save time to meet deadlines, I had to re-use the sprites. If I have time at the end of development I will be able to redesign it, but at the moment this is not necessary as the problem is only very small and is an aesthetic one, not a functional one. They also like the blue background that I implemented instead of the black one.



This code took 2 main attempts to complete.

Reflections:

Objective successfully completed.

Success criteria number 7 - **completed** cooldown adjustment

7	A melee attack the player can use at close range. Time permitting, a cooldown on this weapon.	A basic attack for before better weapons are discovered or bought between levels, to make sure that early levels are playable.

Overview of errors encountered in developing my coded solution

Below is a summary of the errors I encountered in development. **All of these along with their remedial actions taken are explained with justification in detail in the** "testing and errors**" section of the corresponding milestone in the above** "Documenting milestones" **section.**

- Milestone 3: ladders/Climbing - Player would glitch through ground due to not being centred on ladders
- Milestone 4: Enemies - orbs hitting a player and shield or shield and floor at same time would throw up an error as code tries to delete them twice
- Milestone 5: Healthbars and taking damage - Originally no code to stop health going below zero, this created an invalid colour argument as you cannot use an RGB value with a negative number
- Milestone 7: attacking/crouching -Players were able to uncrouch when they shouldn't have been
- Milestone 8: importing level details - Details were not able to be imported from a different folder.
- Milestone 9 : ranged weapons - Aiming was too difficult
- Milestone 10: highscores - Only the top high score was being displayed
- Milestone 12: instructions - the previous background was still showing through
- Milestone 13: polishing features - the weapon needed a cooldown implemented

Evaluation

Testing to inform the evaluation:

In the previous section I tested each module individually. Now I need to check that everything works together as planned (integration testing), and see if it meets my user requirements and criteria to produce the desired effect (alpha testing).

Annotated Usability testing:

- I will use beta testing by getting my stakeholders to play the game and write me a review which is included below. This is my form of usability testing, as the user is the best person to test usability features as they were designed specifically for them.
-THIS IS INCLUDED BELOW IN THE "Usability testing - stakeholder Joe's review after testing"

- Annotating usability features, and discussing their merits and drawbacks
-THIS IS INCLUDED BELOW IN THE "EVALUATING USABILITY FEATURES, JUSTIFYING THEIR SUCCESS, ANNOTATING USABILITY FEATURES"

Post development testing for function and robustness:

- This testing process is accompanied by a video (verbal annotations) to show the testing process, and as evidence of the results of my tests. Video timestamps show the location of these tests. ***The video is titled "Testing to inform evaluation".***

Testing milestone 1: moving/jumping, basic level **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
1	Players can traverse and jump. "A","D", spacebar. Animation should play.	Move according to the assigned keybinds.Jump should look realistic. Walk animation should play.	00:00	Pass
2	Player will not fall through platforms	Player can stand on platform without falling through	00:14	Pass
3	Player cannot walk off-screen	Player should be stuck at the edge, despite movement keys being held	00:28	Pass

Testing milestone 2: menus **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
4	Selection bar moves with input from the arrow keys. Should not move beyond the final options	Yellow bar should move, but not beyond the final options	00:40	Pass
5	Selecting menu options with enter	The *currently selected* option should be performed	01:01	Pass

Testing milestone 3: ladders/climbing **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
6	Player can walk past ladders if desired	Player will continue past the ladder unimpeded	01:26	Pass
7	Players can climb ladders	If “W” held then will go up ladders, if released will start to fall down	01:36	Pass
8	The correct animation is displayed	When climbing, should swap to climbing animation	01:45	Pass

Testing milestone 4: Enemies **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
9	Enemies spawn	At the start of the level the enemies appear	01:55	pass
10	Enemies won't fall off their platforms	Enemies will get to the edge of their platform and turn around	02:02	pass
11	Player can be damaged by enemy	If the player touches an enemy or enemies orb, they will lose health	02:12	pass
12	Enemies can be killed	When attacked with enough range or melee, will die/disappear from screen	02:30	pass

Testing milestone 5: Health bars/taking damage **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
13	Player can take damage	When in lava or touching enemy health decreases	02:38	Fail(lava only damages when moving in it)
14	Health bar updates	Health bar shows damage taken by getting smaller	02:47	pass
15	Player can die	When health bar reaches zero, player dies	02:55	pass

Testing milestone 6: lvl objectives, coins, point system **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
16	Crystals appear and can be	When touching a crystal, it	03:00	pass

	collected	disappears and the counter increments		
17	Level can be completed	When all crystals collected, moves to the next level	03:15	pass
18	Coins appear and can be collected	When touching a coin, it disappears and the counter increments	03:20	pass
19	Enemies can be killed	When enemy killed, counter goes up	03:31	pass
20	At the end of the level, points are tallied	A point breakdown should be shown displaying the players points	03:58	pass

Testing milestone 7: Attacking, crouching **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
21	Player can attack	Can only attack when cooldown is over. Animation should play. Enemy should be damaged	04:05	pass
22	Player can crouch	Player should crouch, display crouch animation, can crawl under 1 block high gaps	04:20	pass
23	Player can uncrouch	Only if allowed(can't uncrouch in a tunnel), then they will be able to uncrouch	04:39	pass

Testing milestone 8: importing level details from text files **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
24	Levels load correctly	When game run, the level loads	04:50	pass

Testing milestone 9: ranged weapons **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
25	Projectiles follow proper path	Should mirror real life, projectiles follow initial path until occupied by gravity	05:16	pass
26	Can only fire projectiles if have ammo	If ammo = 0, then can’t fire projectiles.	05:37	pass

27	Projectiles take effect.	Should damage enemies and destroy any touching blocks.	05:47	pass

Testing milestone 10: Highscores **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
28	High Scores should be displayed	Top 5 scores shown	06:03	pass
29	Display of recently achieved scores	After beating a high score, if you visit the high score screen, the new one will be displayed.	06:18	FAIL
30	Highscores should be displayed in order.	Top scores shown first, and lesser scores later	06:26	pass

Testing milestone 11: Sound effects **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
31	Effects should play	Effects should play at corresponding points(attacking,jumping etc.)	06:59	pass
32	Effects should be at suitable volume levels.	Loud enough to hear, but not enough to be distracting	07:18	pass

Testing milestone 12: Instruction screen **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
33	Instructions displayed	When help option selected, instructions should play	07:29	pass

Testing milestone 13:Polishing features, sword/shield animation **Evidence**

Test no	What's being tested/inputs	Expected output	Timestamp	pass/fail
34	Sword animation displays	Correct animation depending on the direction player is facing	07:48	pass
35	Shield image is displayed	When shield key is pressed, shield image appears	08:12	pass

Evaluation:

Success criteria cross referenced with test evidence.

Evidence of completed success criteria can be found in the “testing to inform evaluation video”, and at the end of each segment of the “documenting milestones section” the completed success criteria are referenced. I use the tests and the criteria to help me evaluate my solution, as seen in the “evaluating the solution using criteria column”

In the evaluation column of the table beneath, I will reference the test numbers of those carried out in the “testing to inform evaluation section” where appropriate. These will be shown in **blue**

I will also reference test numbers of those carried out in the “Considering maintenance issues and limitations:” section beneath, which were some more of my post development tests. These will be shown in **green**

number	Success Criteria	Measure	Evaluating the solution using criteria and **Evidence** **Evidence**
1	Must have WASD for movement, with the addition of spacebar for jump(as w will be for ladders.)	Completed	The keys WASD move the player with w being the key to initiate climbing and spacebar to jump. As seen in the testing video this functions perfectly. With more time and further development, it may be a good idea to include a rebind feature so users can fully customise their experience. **Referencing tests 1,2,3**
2	At least 5 different levels.	Completed	There are 5 unique levels making for interesting gameplay developments. As discovered in my user review, in future I can implement a side scrolling feature to make the levels longer and more interesting for user benefit.
3	Levels increase in difficulty, with one very hard level at the end.	Completed	The final level has many of the most difficult enemies to face, and levels building up to this have subsequently higher numbers of enemies. As discussed in “Increasing the usability of my game” I could instead just increase the stats of the enemy instead of their numbers to achieve the same effect but without crowding the levels which would be better for the user. **Referencing test 1**

4	At least two different types of enemy.	Completed	There is a ranged and melee enemy. Both work exactly as intended and successfully vary levels of challenge to the user. This is evident in both my testing video and user review. **Shown in tests 9,10,11,12**
5	Power ups: at least one.	partially completed	The power up comes in the form of the ranged weapon and so does not fully meet the user requests of power ups like“increased damage sword” that I received in my interviews section. Therefore with this evidence I am marking this criteria as partially completed. Powerup **Shown in tests 25,26,27**
6	A ranged attack that the player can use.	Completed	The ranged weapon comes complete with lines to plot its projectile to make aiming easier for the user and get rid of any potential frustration for them. This as seen in the testing video works exactly as planned. **Shown in tests 25,26,27**
7	A melee attack the player can use at close range. Time permitting, a cooldown on this weapon.	Melee: Completed Cooldown: completed	The melee attack works well, the animation makes it very pleasing for the user,(stated in the user review) **Shown in test 12** The functionality of the cooldown is perfect, it clearly displays the time left so the user can know exactly when to use it. This is tested specifically in my testing section. **Shown in tests 21**
8	Give the player multiple lives or a health bar.	Completed	The player has a health bar that updates in real time to show the damage received by the player, and also changes colour to indicate health left as well. I am very happy with it, the only other feedback I have received is to make it decrease gradually instead of a sudden change when damage is taken. I can implement this in further development to please the user. **Shown in test 14. Visible in all tests**

9	Display lives/health bar.	Completed	The health bar is displayed in the top left of the screen so as not to get in the way of the game. This is shown in my testing video. **Shown in test 14. Visible in all tests**
10	A menu with highscores, level select and play option.	Partially completed	The menu also includes instructions. However I am not marking it fully complete as upon testing I have discovered the bug of a stray projectile being fired.(Whilst this doesn't affect its functionality at all, for me to mark this criteria as completed I would like to get rid of the error. This is simply the case of adding a timer so projectiles can't be fired recently after selecting a level, and will be easily fixed with further dev.) **Shown in tests 4,5**
11	Must have collectable coins in every level.	Completed	Coins are located in various locations throughout the levels, making it a fun challenge for collecting them, so the user will enjoy their experience more, again shown in my testing video. **Shown in test 18, coins are also visible throughout the testing video.**
12	Sound effects and background music. At least 2 different music tracks to reflect difficulty.	Sfx: completed Bg music: fail	Sound effects are included, however I have decided to not include background music for reasons explained in the reflections of my sound effects milestone in "Documenting milestones". These were to avoid all the sounds becoming too hectic for the user **Shown in tests 31,32**.
13	A way to score points by for example defeating enemies (cake soldiers), or collecting coins.	Completed	Points are achieved for killing enemies and collecting coins. Enemies are worth twice as much as coins as they are harder to collect. This rewards players for riskier moves, making the game much more interesting for users to play, how points are achieved are **shown in tests 18,19,20**

14	Have at least three crystals for the player to collect in order to pass the level.	Partially completed	Final level only has one crystal. This is because the challenge on this level is getting past all of the ranged enemies. All of the other levels have 3 crystals. This milestone is deliberately partially completed, as this level design choice benefits the user by providing a clear objective on the final level. **Shown in tests 16,17**
15	Include ladders for vertical traversals too big for the jump feature.	Completed	The ladders work well, and are located in levels with the purpose of providing access to hard to reach points so players can explore and use the whole level, and levels can be more detailed. **Shown in tests 6,7,8**
16	Have at least one damaging block. E.g. lava the player can fall into or spikes they can walk on.	Partially completed	The lava damages the player, however only when the player moves in it or moves the mouse. This is not ideal and in future development damage should be dealt whenever the player is in lava, even if they are stationary. **Shown in test 13**
17	Have many different platforms separated from each other and at different heights.	Completed	Platforms are located around the level to allow traversal to enemies and objectives. They also introduce threats of falling into lava which adds difficulty and immersion to the game, making it a better experience for the user. **Shown throughout testing video**
18	Possibly include a menu to control sound settings, toggles for sound effects on or off etc.	Fail	I have deliberately not implemented this criteria as the only sounds playing are sound effects, which the user can control with their own sound settings on their computers, and so this function is unnecessary.
19	Game must be fun to play for the target audience	8.5/10 according to stakeholders Joe and Charlotte.	I know now that my solution is fun to play, this was a big aim from the start of my project and means that this aspect of it was a success.

Discussing failed tests from the "testing to inform evaluation section"

The failed tests are numbers 13 and 29

Test 13: Players can take damage.
In testing all the eventualities in which the player can be damaged(by an enemy, enemy orb or lava), I noticed a flaw in the lava damaging the player. It only happened when the player was moving in the lava, not just when touching it.
Test 29: Display of recently achieved scores. If a player beats a high score and then goes to the high score screen it will not show until the game is reloaded.

- The solution to test 13 is detailed in the section directly below under **Criteria 16**.
- To fix test 29 I will need to perform more extensive error checking to locate the bug, to do this I will need to make use of IDE features such as trace and step through to allow me to locate exactly where the program goes wrong, from here I will fix the problem, so the user can see their updated highscores without having to reload the game

Addressing unmet criteria with further development:

- **Criteria 16**-Code a function that detects if the player is below a certain y level, if so, cause them to take damage from lava. This is necessary as the collision function doesn't work for some reason if the player is stationary in the lava. I do not know why this is and as such have come up with the solution mentioned above.
- **Criteria 10**-Fix the problem with the stray projectile in the level select. Do this by implementing a timer that doesn't let projectiles be fired until a second after selecting a level.
- **Criteria 5**-Include more power ups like a damage boost and possibly a health boost(in the same vein, health packs can be included)
- **Criteria 3**-Allowing difficulty options to be chosen in the main menu, and having these affect enemy stats.

Usability testing - stakeholder Joe's review after testing

"

The game as a whole has themes resembling platformers like "Super Mario Bros". At the start I am greeted by a vibrant menu screen, this has options to play, select a level, view highscores and instructions. From my short experience playing the game these features all worked wonderfully, and I have no criticisms about the menu itself. However when selecting a level I did notice that it would fire a bullet, I assume this is because the mouse button is bound to the fire command.
The game itself is very fun to play yet also relaxing, the combination of the ranged weapon with its limited ammunition makes the game interesting as it forces you to use it sparingly and also engage in riskier close quarter combat. The animations whilst simple provide a lovely aesthetic touch and make it very visually engaging. With the addition of well suited sound effects, it becomes quite immersive. My only suggestions are to make the levels scroll, so each one lasts longer, and include an option to rebind keys, as I

don't like the fact that the melee button is bound to carriage return. Otherwise I think this is a wonderful platformer.

”

From this I understand that my success criteria have been met. However I have received two pieces of feedback that I can use to further improve my game:

- Fix the level select bug where a projectile is fired
- Implement a side scrolling feature to allow for longer levels
- Include a rebind key feature, or simply change the attack key to one of the mouse buttons.

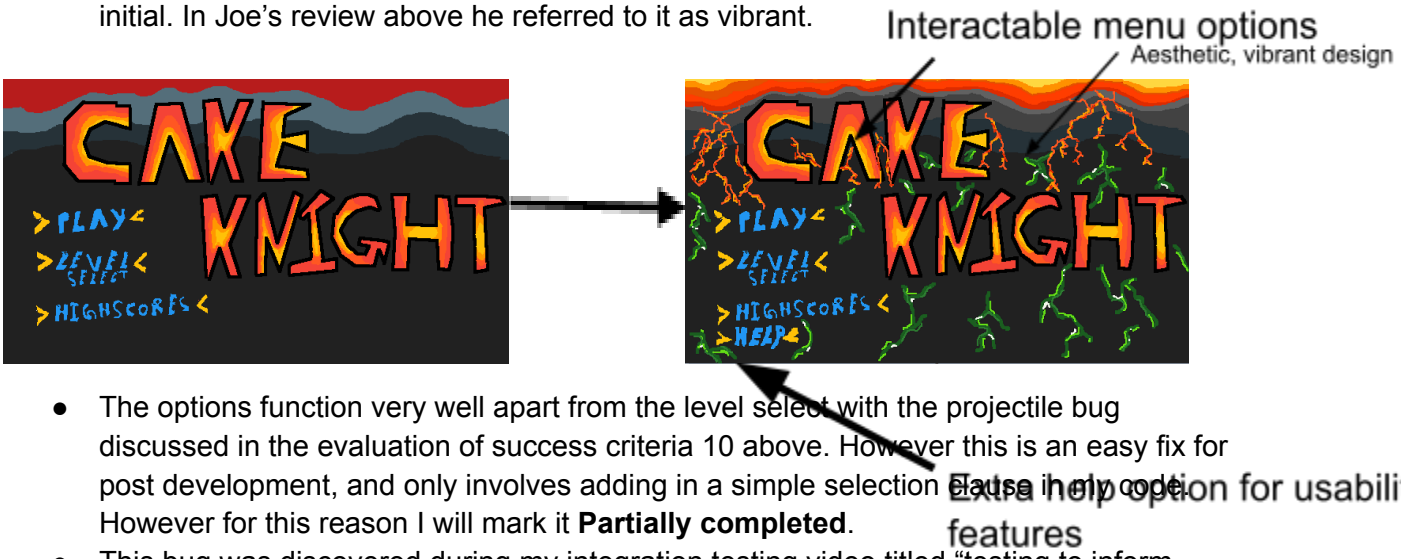
Other future updates could include: Different ammunition types, weapon upgrades and a boss fight, to allow for longer play sessions. These all benefit my users by allowing them to enjoy more features and more detailed levels.

-

<u>Evaluating usability features, justifying their success, Annotating usability features</u>

<u>Welcome/start menu:</u>

- The menu is very aesthetically pleasing. The final design looks much better than the initial. In Joe's review above he referred to it as vibrant.



- The options function very well apart from the level select with the projectile bug discussed in the evaluation of success criteria 10 above. However this is an easy fix for post development, and only involves adding in a simple selection clause in my code. However for this reason I will mark it **Partially completed**.
- This bug was discovered during my integration testing video titled "testing to inform evaluation" and also by my stakeholder in his beta testing.
- Some users would like me to make the menu interactable with the mouse, this would not take too much time to implement, all I need is to store the locations of the options and check if the mouse is over them when clicked. However this is only a cherry on top scenario as all of the functionality is present in the menu feature.

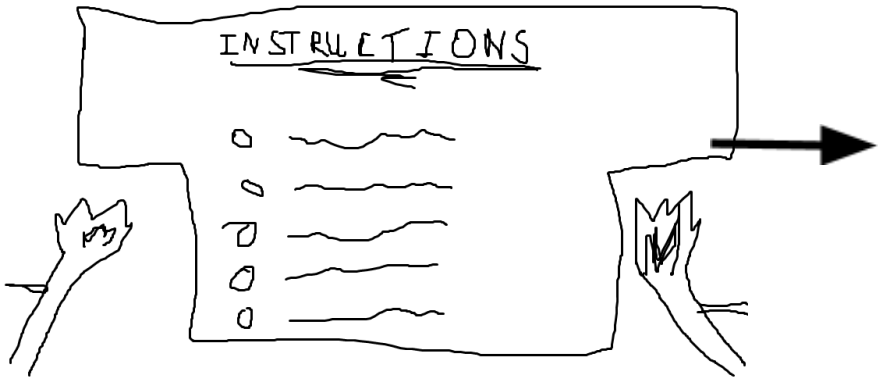
<u>Game screen:</u>

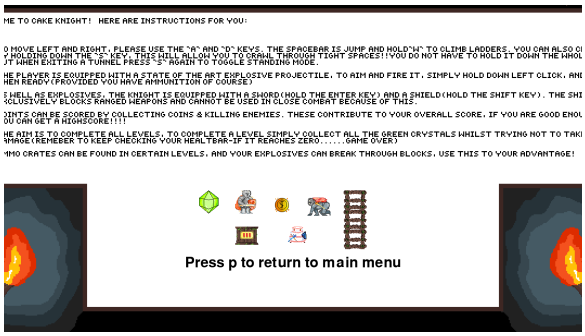
- The HUD and counters displayed all functions very well. They hold up to testing and function as planned. As such my game screen is a great success and is very informative, so I am justified in marking it as **Fully completed**.

Player experiences and difficulty:

- The games steadily get harder as time goes on, however I have received feedback about the wish for a “side scrolling” feature in my section above on usability testing. This could increase the range of difficulty available for levels as it allows for better level design. Whilst this section is a success in terms of my initial criteria, it could be improved upon to result in a better experience for the user, and as such I will mark it as **Fully completed**, but with room for improvement which I discuss in the section beneath this titled “Ideas on increasing the usability of my game”

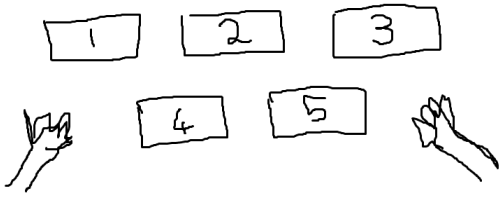
Instruction screen:





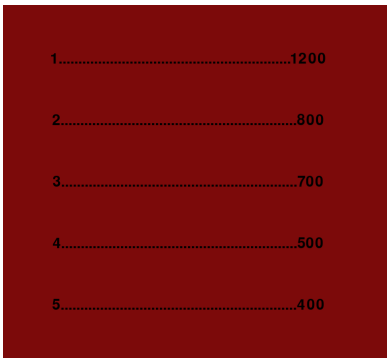
- The instruction screen has progressed well, it has a nice aesthetic design to appeal to the users, in keeping with the cave/torch theme from the level select
- Charlotte has told me “The instructions are clear, and the option to return to the main menu is made very clear so I don't have to search for it”
- From these points I will mark this usability feature as **completed**

Level select screen



- I am happy with the way the design has progressed. However in my design of this usability feature, I mentioned I would like a level preview in each box, this is because I thought it would be a great idea to show the user the level beforehand, as an indication of the difficulty so they can choose what is best for them. However due to time constraints I was not able to complete this, for this reason I will be marking this feature as a **partial success**, as while all the functionality works, it does not meet my initial design/aesthetic requirements.

HighScore Screen:



- The high score screen is very bland. The user feedback from my implementation section was that this could use some work. I would have liked to add some design elements like a trophy, or even some vines.
- The functionality is all there, apart from the small error discussed in Test 29 of the testing video, timestamp 06:18 .

Due to the lack of design aesthetics (which my stakeholder requested),(this was because of time constraints, I couldn't do the pixel art in time), and the small bug I will mark this feature as a **partial success**, as the highscores will update when the game is re-launched.

Ideas on increasing the usability of my game with further development

I would also like to increase the usability to help me reach a wider audience. I have some ideas on how to achieve this.

- Allow for a two player option. I wanted to do this originally and my stakeholders requested this, however I was unable due to time constraints as mentioned in my analysis section.

- For this to work it would be a good idea for me to include another menu option to allow for keybinds so the users can choose the controls which suit them, making playing the game easier as they won't have to crowd over the same area of a keyboard.
- Player classes. With a multiplayer mode, I can also introduce characters with unique capabilities such as higher defence or movement speed, this will make the game more interesting, and is a huge hit in one of the games I researched for my analysis-"castle crashers".
- At the moment to increase the difficulty I simply add more enemies and more of the ranged enemies. However it would be easy to add a difficulty attribute in the enemy class which will adjust its health and damage in game. This will allow for a different method of varying the difficulty, a menu option at the start of the game, therefore increasing the usability for players with different skill levels.

Considering Maintenance issues and Limitations:

Maintenance needed is identified using tests performed with test tables created in "Further data for post development" in the design section. These tests along with the results are displayed below for reference.

Limitations are identified and whilst there are not many, there are:

- Minor issue of availability: The game can only be played on computers
- The game is only single player(This is elaborated on in more detail in the "Ideas on increasing the usability of my game" section.)
- Currently there is no option to load saved games

no	Post Development Test	Testing to be done	Result
1	Level difficulty increases	More enemies each level? Stronger/ better positioned enemies each level? Do the levels seem a suitable difficulty- get stakeholders to test this for me.	Pass. When my stakeholders played the game they were satisfied with the rate of difficulty increase.
2	Easy to use the projectiles	Does the projectile path feel intuitive? Is acceleration due to gravity too great?	Pass. With the addition of the path plotters, aiming is easy. It took some fiddling with values but the g-value works well.
3	Weapons are adequately balanced.	Does the sword/ projectile deal enough damage? Are enemies too hard to kill?	Fail. The sword deals too much damage, kills enemies too easily
4	Hitboxes are an	Do the enemies take damage	Pass. Hitboxes are a suitable size,

	adequate size.	when the sword/ projectile image appears to collide, not when the underlying rect objects actually collide.	they only take damage when they look to be hit.

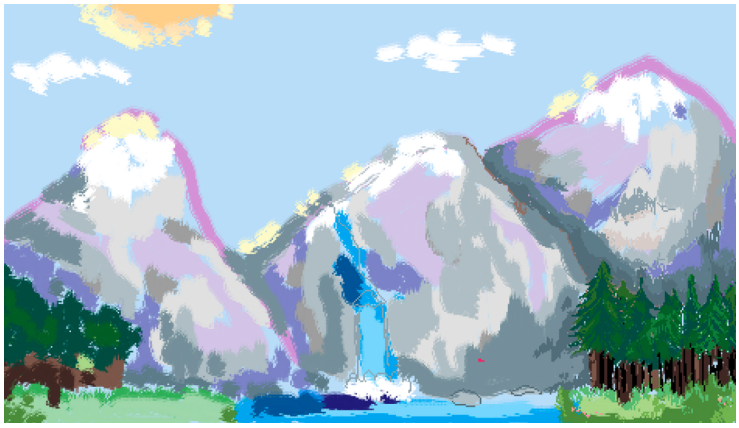
- From this testing, I understand that the sword needs to deal less damage.
- I also have noticed that the lava makes a lot of noise, this is because it damages the player when they move through it, and therefore the damage sound effect is played frequently.
- A last thing I would like to add for further development would be a shield animation that utilises the image of the shield the player naturally carries, rather than the extra blue shield image. This adds no functionality but I believe it would be aesthetically pleasing to the user.

Developing program to deal with Limitations and Maintenance

- The shield animation and the lava sound can be easily remedied. The shield animation just needs the graphics drawing and then I can implement it the same way I do with all of my other animations like climbing and crouching.
- To fix the lava I will remove the damage sound effect when being damaged by lava or add a quieter more pleasant sound that can be played continuously with no adverse effect like a sizzle.
- Maintaining my program will not be difficult, any complex functions are commented, and code is grouped together in sections (all the movement code is in one place in the main while loop for example, the same for taking and dealing damage ect.). This makes it easier to follow and add to. Its modular structure also makes it easy to debug as units are isolated from one another in a sense.
- My program does not have many limitations, the solution works almost exactly as intended with the exception of a few minor changes for the better, and a few bugs. However it is only available for use on certain computers. To try and increase availability, in the future I could create a version that is able to operate on mobile phones, so my game is even more portable and quick to use(which is an important factor I mention in my analysis). Therefore to get rid of a small limitation of availability, I can implement this.
- In order to implement the load saved data feature mentioned above in “Considering maintenance and limitations” all I would need to do is code a function to write the current game data to a file. And code one more to load this when the user tries to load a saved file. This game data would include current level, coins collected, enemies killed, ammo left ect.

Given further time in the development section, I would have liked to have designed a set of backgrounds for my game, instead of just the monotone light blue background that I am using at

the moment. I had an initial sketch of a background that I'm including below as a foundation to build on in further development.



<u>Does the project meet my requirements, key ideas I have noted/ overall review.</u>

Most of the tests in my post development and informing evaluation test sections were successes. I find this is because most errors were encountered during extensive testing of my milestones in the “Documenting milestones” segment within the “Developing coded solution ”section. As such I can see that this approach is very successful in developing coded projects, as any remaining errors are often located with post development tests and can be easily ironed out with quick fixes.

In general I am very happy with my solution. I find it fits my problem that I outlined in the “analysis” section. Below I have quoted what I originally wrote in my “Description of the problem” segment in the analysis section, to show how well my final solution fits the initial requirements.

- *“Many people yearn for the simplicity of a game with minimal story line, that they do not need to invest hours in for any result, that is fun to play, simple to use and master, whilst still retaining a significant level of difficulty to keep them entertained”*

My game’s storyline, if it could even be said to have one, is not complex. All it consists of is a knight collecting coins, crystals and killing cake enemies.
It is very simple to master, due to its limited number of controls and as shown above in my evaluation, the difficulty settings are optimal. Therefore this initial design requirement from the analysis section has been met

- *“In this project I aim to modernise the original games like DK or Super Mario Bros, with better graphics and animations, now possible with modern computing power. I will still try and keep the classic feel, with pixelated animations and characters but will update the quality of them. I will expand on the functionality*

and usability of the original platformers by creating numerous levels, of varying difficulty, so the player can choose specifically the level of challenge they want"

As written by one of my stakeholders in "Usability testing - stakeholder Joe's review after testing" segment, *"The game as a whole has themes resembling platformers like 'Super Mario Bros'."* The level select feature also perfectly allows the user to choose the level of challenge they want. Therefore this initial design requirement from the analysis section has been met.

I believe that the graphics work wonderfully, and the game is very fun to play, as well as being a great hit with my stakeholders and therefore also target audience. Almost all features work as planned and any limitations, maintenance, unmet success criteria and ideas to increase usability as discussed in the sections above can be very easily met in further development using the methods outlined.
I believe that my project has been a great success, and this is evident through my evaluation sections and stakeholder reviews.

Bibliography/references:

The only material throughout this project that is not my own property are the sound effects I created online. All graphics I made myself with the help of a pixel art site

Sound effects creator site: https://sfxr.me/
Pixel art site I used(*to make my own graphics*) : https://www.pixilart.com/

Pygame library that I have utilised: https://www.pygame.org/docs/

Final code listing:

```
import math
import random
import pygame,sys
from pygame import mixer
clock = pygame.time.Clock()
from pygame.locals import *
pygame.init()

pygame.display.set_caption("Cake Knight")

    screenWidth = 1920
    screenHeight = 1080

    win = pygame.display.set_mode((screenWidth,screenHeight))

    red = (255,0,0)
    green = (0,255,0)
```

```
18.   blue = (0,0,255)
19.   white = (255,255,255)
20.   black = (0,0,0)
21.   gold = (245,188,66)
22.   bloodred = (124,10,10)
23.   victoryGold = (255, 204, 0)
24.
25.
26.   font1 = 'freesansbold.ttf'
27.   pygame.mixer.music.load("soundfx/damageSound.wav")
28.   damageSound = pygame.mixer.Sound("soundfx/damageSound.wav")
29.   pygame.mixer.music.load("soundfx/explosionSound.wav")
30.   explosionSound =
    pygame.mixer.Sound("soundfx/explosionSound.wav")
31.   pygame.mixer.music.load("soundfx/coinSound.wav")
32.   coinSound = pygame.mixer.Sound("soundfx/coinSound.wav")
33.   pygame.mixer.music.load("soundfx/jumpSound.wav")
34.   jumpSound = pygame.mixer.Sound("soundfx/jumpSound.wav")
35.   pygame.mixer.music.load("soundfx/attackSound.wav")
36.   attackSound = pygame.mixer.Sound("soundfx/attackSound.wav")
37.   pygame.mixer.music.load("soundfx/shieldSound.wav")
38.   shieldSound = pygame.mixer.Sound("soundfx/shieldSound.wav")
39.   attackImage1R =
    pygame.image.load("Images/attackImage1Right.png").convert_alpha
    ()
40.   attackImage2R =
    pygame.image.load("Images/attackImage2Right.png").convert_alpha
    ()
41.   attackImage3R =
    pygame.image.load("Images/attackImage3Right.png").convert_alpha
    ()
42.   attackImage4R =
    pygame.image.load("Images/attackImage4Right.png").convert_alpha
    ()
43.   attackAnimationRight =
    [attackImage1R,attackImage2R,attackImage3R,attackImage4R]
44.   attackImage1L =
    pygame.image.load("Images/attackImage1Left.png").convert_alpha(
    )
45.   attackImage2L =
    pygame.image.load("Images/attackImage2Left.png").convert_alpha(
    )
46.   attackImage3L =
    pygame.image.load("Images/attackImage3Left.png").convert_alpha(
    )
```

```
47.   attackImage4L =
   pygame.image.load("Images/attackImage4Left.png").convert_alpha(
   )
48.   attackAnimationLeft =
   [attackImage1L,attackImage2L,attackImage3L,attackImage4L]
49.   instructionBackgroundImage =
   pygame.image.load("Images/instructionBackground.png").convert_a
   lpha()
50.   knightDeadImage =
   pygame.image.load("Images/knightDead.png").convert_alpha()
51.   shieldImageLeft = pygame.image.load("Images/shield
   left.png").convert_alpha()
52.   shieldImageRight = pygame.image.load("Images/shield
   right.png").convert_alpha()
53.   orbImage =
   pygame.image.load("Images/orbImage.png").convert_alpha()
54.   outsideCloudImage =
   pygame.image.load("Images/outsideCloud.png").convert_alpha()
55.   caveCloudImage =
   pygame.image.load("Images/caveCloud.png").convert_alpha()
56.   cakeEnemy =
   pygame.image.load("Images/attackerEnemy.png").convert_alpha()
57.   cakeEnemyHigh =
   pygame.image.load("Images/attackerEnemyHigh.png").convert_alpha
   ()
58.   cakeEnemyMedium =
   pygame.image.load("Images/attackerEnemyMedium.png").convert_alp
   ha()
59.   cakeEnemyLow =
   pygame.image.load("Images/attackerEnemyLow.png").convert_alpha(
   )
60.   cakeAttackerImages =
   [cakeEnemy,cakeEnemyHigh,cakeEnemyMedium,cakeEnemyLow]
61.   levelPreviewImageFiller
   =pygame.image.load("Images/levelPreviewImageFiller.png").conver
   t_alpha()
62.   bombImage =
   pygame.image.load("Images/bombImage.png").convert_alpha()
63.   levelSelectBg =
   pygame.image.load("Images/levelSelectBg.png").convert_alpha()
64.   healthbar =
   pygame.image.load("Images/healthbar.png").convert_alpha()
65.   ammoCounterImage =
   pygame.image.load("Images/ammoCounter.png").convert_alpha()
```

```
66.   timesSymbolImage =
   pygame.image.load("Images/timesSymbol.png").convert_alpha()
67.   slashSymbolImage =
   pygame.image.load("Images/slashSymbol.png").convert_alpha()
68.   image0 =
   pygame.image.load("Images/image0.png").convert_alpha()
69.   image1 =
   pygame.image.load("Images/image1.png").convert_alpha()
70.   image2 =
   pygame.image.load("Images/image2.png").convert_alpha()
71.   image3 =
   pygame.image.load("Images/image3.png").convert_alpha()
72.   image4 =
   pygame.image.load("Images/image4.png").convert_alpha()
73.   image5 =
   pygame.image.load("Images/image5.png").convert_alpha()
74.   numbers0to5 = [image0,image1,image2,image3,image4,image5]
75.   crystalImage =
   pygame.image.load("Images/crystal.png").convert_alpha()
76.   dirt =
   pygame.image.load("Images/dirtRectangle.png").convert_alpha()
77.   grass =
   pygame.image.load("Images/grassRectangle.png").convert_alpha()
78.   cobble =
   pygame.image.load("Images/cobbleRectangle.png").convert_alpha()
79.   grassCobble =
   pygame.image.load("Images/grassyCobbleRectangle.png").convert_a
   lpha()
80.   lavaSurface =
   pygame.image.load("Images/lavaRectangleSurface.png").convert_al
   pha()
81.   lava =
   pygame.image.load("Images/lavaRectangle.png").convert_alpha()
82.   ammoCrateImage =
   pygame.image.load("Images/ammoCrate.png").convert_alpha()
83.   outsideBg =
   pygame.image.load("Images/outsideBg.png").convert_alpha()
84.   splashScreenImg = pygame.image.load("Images/splash screen
   v3.png").convert_alpha()
85.   ladderImage =
   pygame.image.load("Images/ladder.png").convert_alpha()
86.   climbing1 =
   pygame.image.load("Images/climbing1.png").convert_alpha()
87.   climbing2 =
   pygame.image.load("Images/climbing2.png").convert_alpha()
```

```
88.   climbingImages = [climbing1,climbing2]
89.   knightRight =
   pygame.image.load("Images/knightRight.png").convert_alpha()
90.   knightLeft =
   pygame.image.load("Images/knightLeft.png").convert_alpha()
91.   right1 =
   pygame.image.load("Images/runningRight1.png").convert_alpha()
92.   right2 =
   pygame.image.load("Images/runningRight2.png").convert_alpha()
93.   right3 =
   pygame.image.load("Images/runningRight3.png").convert_alpha()
94.   left1 =
   pygame.image.load("Images/runningLeft1.png").convert_alpha()
95.   left2 =
   pygame.image.load("Images/runningLeft2.png").convert_alpha()
96.   left3 =
   pygame.image.load("Images/runningLeft3.png").convert_alpha()
97.   crouchRight1
   =pygame.image.load("Images/crouchRight1.png").convert_alpha()
98.   crouchLeft1 =
   pygame.image.load("Images/crouchLeft1.png").convert_alpha()
99.   crouchRight2
   =pygame.image.load("Images/crouchRight2.png").convert_alpha()
100.  crouchLeft2 =
   pygame.image.load("Images/crouchLeft2.png").convert_alpha()
101.  crouchingRight =[crouchRight1,crouchRight2]
102.  crouchingLeft =[crouchLeft1,crouchLeft2]
103.  bigCoinImage =
   pygame.image.load("Images/bigCoinImage.png").convert_alpha()
104.  coin1 = pygame.image.load("Images/coin1.png").convert_alpha()
105.  coin2 = pygame.image.load("Images/coin2.png").convert_alpha()
106.  coinImages = [coin1,coin2]
107.  runningRight =[right1,right2,right3]
108.  runningLeft =[left1,left2,left3]
109.  playerImage = knightRight
110.  damageCounter = 0
111.  playerHealth = 250
112.  healthTimer = 0
113.  playerMoney = 0
114.  coinValue = 1
115.  playerAmmo = 5
116.  enemiesKilled = 0
117.
118.  TILE_SIZEX = 64
119.  TILE_SIZEY = 54
```

```
120.
121.  crouchCycle = 0
122.  walkCycle= 0
123.  climbingCycle = 0
124.  attackCycle = 0
125.  coinCounter = 0
126.  shotsFiredCounter = 0
127.  rangedEnemyFireCounter = 0
128.  oneTime = True
129.  oneTimeLevelConstructor = True
130.  oneTimeScoreCalculation = True
131.  radiusAimCircle = 0
132.  playerMaxAmmo = 5
133.  aimDeltax = 0
134.  aimDeltay = 0
135.  aimLineLength = 0
136.  aimCircleCenter = [0,0]
137.  aimVectorDeltax = 0
138.  aimVectorDeltay =  0
139.  aimLineEndPoint = [0,0]
140.  totalScore = 0
141.  weaponCooldown = 250
142.
143.  enemyCollided = ""
144.  tileCollided = ""
145.  isCollidedTile = False
146.  isCollidedEnemy = False
147.  crouching = False
148.  moving_right = False
149.  moving_left = False
150.  climbing = False
151.  canClimb = False
152.  player_y_vel = 0
153.  air_timer = 0
154.  isJump = False
155.  isPlayerAttacking = False
156.  aiming = False
157.  aimed = False
158.  shielding = False
159.
160.  isPlayerHit= False
161.  playerKnockbackCounter = 0
162.  damagedSide = ""#so you know what side the player hits the
      neemy with
```

```
player_rect = pygame.Rect(50, 50, 64, 64)#starting co-ords, players width and height
test_rect = pygame.Rect(100,100,100,50)
player_movement = [0,0]
playerDealDamageCounter = 0
sword = pygame.Rect(-100,-100,76,7)
shield = pygame.Rect(-200,-200,9,60)
currentLevel = 1
highScores = False
levelSelect = False
game = False
menu = True
instructions = False
options = ["play","levelSelect","highScores","intsructions"]# variables for the splash screen
choice = 0  # Choice holds the mode they chose, level select,play ect
selected = False
lastFacing = "1"# so when he isnt moving i can put the correct facing standing image- right or left
cantCollideList = [" ",".","l","h","c","m","e","a"]
bombCantHitList = [" ","h","c","m","o","p"]
damageBlocks = []#blocks that deal damage to the player
ladders = []
coins = []
enemies = []
bombs = []
orbs = []
scores = []
totalCrystals = 0
crystalsGathered = 0
crystals = []
ammoCrates = []
topFiveScores = []

levelsAndNames=[
[1,"level1.txt"],
[2,"level2.txt"],
[3,"level3.txt"],
[4,"level4.txt"],
[5,"level5.txt"]
]

level1 = [" "]*20
level2 = [" "]*20
```

```
204.  level3 = [" "]*20
205.  level4 = [" "]*20
206.  level5 = [" "]*20
207.  for i in range(20):#sets up all the level arrays
208.     level1[i] = [" "]*32
209.  for i in range(20):
210.     level2[i] = [" "]*32
211.  for i in range(20):
212.     level3[i] = [" "]*32
213.  for i in range(20):
214.     level4[i] = [" "]*32
215.  for i in range(20):
216.     level5[i] = [" "]*32
217.  levels = [level1,level2,level3,level4,level5]
218.
219.
220.
221.
222.  def test():
223.     print("Test")
224.
225.
226.
227.  class Enemy():
228.     def
   __init__(self,x,y,width,height,health,damage,category,movementx
   =3,movementy=0,facing=1,counter=0,fireCounter = 0):
229.         self.x = x
230.         self.y = y
231.         self.width = width
232.         self.height = height
233.         self.health = health
234.         self.damage = damage
235.         self.category = category
236.         self.movementx = movementx
237.         self.movementy =  movementy
238.         self.facing = facing
239.         self.counter = counter
240.         self.totalHealth = health
241.         self.fireCounter = 0
242.
243.         self.rect =
   pygame.Rect(self.x,self.y,self.width,self.height)
244.     def draw(self):
245.         imageNumber = self.totalHealth/self.health
```

```
246.         remainder  = self.totalHealth % self.health
247.         if remainder !=0:
248.             if remainder>=0.5:#rounds up
249.                 imageNumber = int(round(imageNumber))
250.             else:#rounds down
251.                 imageNumber = int(imageNumber - remainder)
252.         else:
253.             imageNumber = int(imageNumber)
254.         if imageNumber >3:
255.             imageNumber = 4
256.         pygame.draw.rect(win,red,self.rect,-1)
257.
    win.blit((cakeAttackerImages[imageNumber-1]),(self.rect[0],self
    .rect[1]))
258.
259.     def move(self):
260.         self.rect, collisionsEnemy = move(self.rect,
    (self.movementx,self.movementy), tile_rects)#updates the enemy
    rect, retrieves a list of things it collides with
261.         if collisionsEnemy["bottom"]:
262.             self.movementy = 0
263.             self.counter = 0
264.         if  not (collisionsEnemy["bottom"]):#if not standing
    on block after movement, undoes the movement and turns the enemy
    around
265.             self.counter +=1
266.             if self.counter ==3:
267.                 self.rect[1]-= 1.2
268.                 self.rect[0]-= self.movementx
269.                 self.movementx = self.movementx*-1
270.             if self.movementx>0:
271.                 self.facing = 1
272.             else:
273.                 self.facing = -1
274.         if (collisionsEnemy["right"]) or
    (collisionsEnemy["left"]):#if hits something to the right turns
    around, no need to reverse movement, this for horizontals is done
    within move function
275.             self.movementx=self.movementx*-1
276.             self.facing = self.facing * -1
277.
278.     def attack(self):
279.         if self.fireCounter == 0:
280.
    orbs.append(Orb(15,self.x,self.y,player_rect,100,10,orbImage))
```

```
281.
282.
283.            self.fireCounter += 1
284.            if self.fireCounter >200:
285.                self.fireCounter = 0
286.
287.    class Orb():
288.        def
       __init__(self,length,enemyx,enemyy,player_rect,damage,vel,image
       ,collisionTile = " ", collisionPlayer= " "):
289.            self.damage = damage
290.            self.image = image
291.            self.length = length
292.            self.collisionTile = collisionTile
293.            self.collisionPlayer = collisionPlayer
294.            self.x = enemyx
295.            self.y = enemyy
296.            self.vel = vel
297.            self.startx = enemyx
298.            self.starty = enemyy
299.            self.calculateVectorOneTime = True
300.        def draw(self):
301.            self.rect =
       pygame.draw.rect(win,red,(self.x,self.y,self.length,self.length
       ))
302.            win.blit(self.image,(self.x,self.y))
303.
304.        def move(self,player_rect):
305.
306.
307.            deltax = self.startx -
       (player_rect[0]+player_rect[2]//2)
308.            deltay = (player_rect[1]+player_rect[3]//2) -
       self.starty#calculates the difference in the players position and
       the orbs starting position
309.            displacement = (deltax**2 + deltay**2)**0.5
310.
311.            self.velx = (deltax/displacement) * self.vel
312.            self.vely = (deltay/displacement) * self.vel
313.
314.            if self.startx>player_rect[0] and self.velx>0:#makes
       sure that the velocities have the correct direction
315.                self.velx = self.velx*-1
316.            elif self.startx<player_rect[0] and self.velx<0:
317.                self.velx = self.velx*-1
```

```
318.         if self.starty>player_rect[1] and self.vely>0:
319.             self.vely = self.vely*-1
320.
321.         self.x += self.velx
322.         self.y += self.vely
323.
324.
325.
326.
327.  class Bomb():
328.      def
     __init__(self,length,player_rect,damage,velx,vely,image,gravity
     =0.1,collisionTile = " ",collisionEnemy=" "):
329.         self.damage = damage
330.         self.collisionEnemy = collisionEnemy
331.         self.collisionTile = collisionTile
332.         self.length = length
333.         self.image = image
334.         self.collided = False
335.         self.collidedEnemy = False
336.         self.collidedTile = False
337.         self.gravity = gravity
338.         self.x = player_rect[0] + (player_rect[2]//2)#sets
     starting co-ords of profectile to center of player
339.         self.y = player_rect[1] + (player_rect[3]//2)#
340.
341.
342.         self.velx = velx//25#divide by 50 otherwise too big
343.         self.vely = vely//25
344.         #stops the velocity from being too small
345.         if abs(self.velx)<1:
346.             if self.velx>=0:
347.                 self.velx = 1
348.             else:
349.                 self.velx = -1
350.         if abs(self.vely)<1:
351.             if self.vely>=0:
352.                 self.vely = 1
353.             else:
354.                 self.vely = -1
355.
356.
357.     def draw(self):
```

```
358.         self.rect =
   pygame.draw.rect(win,red,(self.x,self.y,self.length,self.length
   ),1)
359.         win.blit(self.image,(self.x,self.y))
360.     def move(self):
361.
362.         self.x+= round(self.velx)
363.         self.y += round(self.vely)
364.
365.         self.vely += self.gravity # the gravity on the
   projectile
366.         if self.vely>5:
367.             self.vely = 5
368.     def collisionCheck(self,tiles):
369.         for tile in tiles:
370.             if self.rect.colliderect(tile[0]):
371.                 self.collidedTile = True
372.                 self.collisionTile = tile
373.         for enemy in enemies:
374.             if self.rect.colliderect(enemy):
375.                 self.collidedEnemy = True
376.                 self.collisionEnemy = enemy
377.
378.         return
   (self.collidedTile,self.collidedEnemy,self.collisionTile,self.c
   ollisionEnemy)
379.
380.
381. def drawWeaponCooldown(weaponCooldown):
382.     colour = (50,50,(weaponCooldown))
383.     pygame.draw.rect(win,colour,(450,5,weaponCooldown,60))
384.     win.blit(healthbar,(400,0))
385.
386.
387.
388.
389. def gameOver(totalScore):
390.     totalScore += playerMoney*50 + enemiesKilled*100
391.     win.fill(bloodred)
392.     draw_text(font1,"Game ",black,50,900,540)
393.     pygame.display.update()
394.     pygame.time.wait(750)
395.     draw_text(font1," Over",black,50,1050,540)
396.     pygame.display.update()
397.     pygame.time.wait(750)
```

```
398.     draw_text(font1,("Total Score:  " +
   str(totalScore)),white,30,970,585)
399.     pygame.display.update()
400.     pygame.time.wait(5000)
401.     game = False
402.     menu = True
403.     return(menu,game,totalScore)
404.
405.  def victory():
406.     win.fill(victoryGold)
407.
   draw_text(font1,"Victory!",black,60,screenWidth//2,screenHeight
   //2)
408.     draw_text(font1,("Total Score:  " +
   str(totalScore)),white,30,screenWidth//2,screenHeight//2+150)
409.     pygame.display.update()
410.     pygame.time.wait(2500)
411.     game = False
412.     menu = True
413.     file = open("highscores/highscores.txt","a")
414.     file.write("\n")
415.     file.write(str(totalScore)+",")
416.     file.close()
417.     return(menu,game,totalScore)
418.
419.
420.
421.  def importLevel(levelNumber):
422.     levelToLoad = ""
423.     for i in range(len(levelsAndNames)):
424.         if levelsAndNames[i][0] == levelNumber:
425.             levelToLoad = levelsAndNames[i][1]
426.     file = open(("leveldetails/" + levelToLoad),"r")
427.     x = 0
428.     for line in file:
429.         content = line.split(",")
430.         for i in range(len(content)-1):
431.             (levels[levelNumber-1])[x][i] = content[i]
432.         x +=1
433.     return(levels[levelNumber-1])
434.
435.
436.  def canStand():
437.     global isCanStand
```

```
438.     isCanStand = True# holds boolean value representing if
   player can go off crouch or not
439.     hitList = []
440.     player_rect[3] = 64#sets player to non crouch position
441.     player_rect[1] = player_rect[1]-14
442.     playerTop = player_rect[1]
443.     playerBottom = player_rect[1]+ player_rect[3]
444.     tileBottom = 0# initialises the variable
445.     for tile in tile_rects:
446.         if player_rect.colliderect(tile):
447.             hitList.append(tile)
448.
449.     for tile in hitList:#if the player is touching any tile
450.         tileBottom = tile[1]+tile[3]
451.         if playerTop<=tileBottom: # if the players head is
   above the bottom of the tile
452.             if playerBottom>tileBottom:# if the players bottom
   is below the bottom of the tile
453.                 isCanStand = False#then the player cant stand
   up
454.                 player_rect[3]=54
455.                 player_rect[1]= player_rect[1]+14
456.     return isCanStand
457.
458.
459.
460.
461. def resetValues():
462.     enemiesKilled = 0
463.     playerAmmo = playerMaxAmmo
464.     crystalsGathered = 0
465.     oneTimeLevelConstructor = True
466.     moving_right = False
467.     moving_left = False
468.     climbing = False
469.     crystals=[]
470.     coins=[]
471.     totalCrystals = 0
472.     player_rect.x = 10
473.     player_rect.y = 10
474.     playerHealth = 250
475.     playerMoney = 0
476.
   return(crystalsGathered,oneTimeLevelConstructor,moving_right,mov
```

```
    ing_left,climbing,crystals,coins,totalCrystals,player_rect.x,pl
    ayer_rect.y,playerHealth,playerMoney)
477.
478.  def endLevelScores(totalScore):
479.      text1 = "Coins Collected:  " + str(playerMoney)
480.      text2 = "Enemies Defeated:  " + str(enemiesKilled)
481.      text3 = "Total Score:  " + str(playerMoney*50 +
    enemiesKilled*100)
482.      totalScore += playerMoney*50 + enemiesKilled*100
483.      draw_text(font1,text1,white,30,970,540)
484.      draw_text(font1,text2,white,30,970,585)
485.      draw_text(font1,text3,white,40,970,650)
486.      return totalScore
487.
488.  def draw_text(font,text,text_colour,size,xcor,ycor):
489.
490.      fonts = pygame.font.Font(font,size)
491.      text_surface = fonts.render(text,True,text_colour)
492.      text_rect = text_surface.get_rect()
493.      text_rect.center = (xcor,ycor)
494.      win.blit(text_surface,text_rect)
495.
496.  def displayMoneyValue():
497.      draw_text(font1,str(playerMoney),red,40,(1500),(150))
498.
499.  def drawHealthBar():
500.
501.
    pygame.draw.rect(win,(255,(playerHealth),0),(100,5,playerHealth
    ,60))#health bar will go more red as player gets lower health
502.      win.blit(healthbar,(50,0))
503.
504.  def drawAmmoCounter(playerAmmo):
505.      win.blit(ammoCounterImage,(1500,0))
506.      win.blit(timesSymbolImage,(1560,0))
507.      win.blit((numbers0to5[playerAmmo]),(1620,0))
508.
509.  def displayMoneyCounter(playerMoney):
510.      win.blit(bigCoinImage,(1700,0))
511.      win.blit(timesSymbolImage,(1760,0))
512.      win.blit((numbers0to5[playerMoney//coinValue]),(1820,0))
513.
514.  def displayKillCounter(enemiesKilled):
515.      win.blit(cakeEnemy,(1300,7))
516.      win.blit(timesSymbolImage,(1360,0))
```

```
517.     win.blit((numbers0to5[enemiesKilled]),(1420,0))
518.
519.  def displayCrystalsGathered(totalCrystals,crystalsGathered):
520.     win.blit(crystalImage,(1040,0))
521.     win.blit(numbers0to5[totalCrystals],(1100,0))
522.     win.blit(slashSymbolImage,(1160,0))
523.     win.blit(numbers0to5[crystalsGathered],(1220,0))
524.
525.  def collision_test(rect,tiles):
526.     hit_list = []
527.     for tile in tiles:
528.       if rect.colliderect(tile):
529.          hit_list.append(tile)
530.     return hit_list
531.
532.  def move(rect, movement, tiles):
533.     collision_types = {'top': False, 'bottom': False, 'right':
   False, 'left': False}
534.     rect.x += movement[0]
535.     hit_list = collision_test(rect, tiles)
536.     for tile in hit_list:
537.         if movement[0] > 0:
538.             rect.right = tile.left
539.             collision_types['right'] = True
540.         elif movement[0] < 0:
541.             rect.left = tile.right
542.             collision_types['left'] = True
543.     rect.y += movement[1]
544.     hit_list = collision_test(rect, tiles)
545.     for tile in hit_list:
546.         if movement[1] > 0:
547.             rect.bottom = tile.top
548.             collision_types['bottom'] = True
549.         elif movement[1] < 0:
550.             rect.top = tile.bottom
551.             collision_types['top'] = True
552.     return rect, collision_types
553.
554.
555.  level1 = importLevel(1)
556.  level2 = importLevel(2)
557.  level3 = importLevel(3)
558.  level4 = importLevel(4)
559.  level5 = importLevel(5)
560.
```

```
561. levels = [level1,level2,level3,level4,level5]
562. game_map = level1
563.
564.
565. run = True
566. while run:
567.
568.     while menu:
569.         oneTimeScoreCalculation = True
570.         win.blit(splashScreenImg,(0,0))
571.         for event in pygame.event.get():
572.             if event.type == KEYDOWN:#can press the arrow keys
    to move between options
573.                 if event.key == K_DOWN:
574.                     if choice < len(options)-1:
575.                         choice +=1
576.                         print(options[choice])
577.                 elif event.key ==K_UP:
578.                     if choice >0:
579.                         choice -=1
580.                         print(options[choice])
581.                 elif event.key == K_RETURN:#when enter is
    pressed option is finalised
582.                     selected = True
583.                     menu = False
584.                     print("Final choice",options[choice])
585.
586.         if choice == 0:
587.             LinePos1 =(100,(600+choice*200))
588.             LinePos2 =(700,(600+choice*200))
589.         if choice == 1:
590.             LinePos1 =(100,(600+choice*200))
591.             LinePos2 =(700,(600+choice*200))
592.         if choice == 2:
593.             LinePos1 =(100,(515+choice*200))
594.             LinePos2 =(700,(515+choice*200))
595.         if choice == 3:
596.             LinePos1 =(100,(1010))
597.             LinePos2 =(700,(1010))
598.
599.
    pygame.draw.line(win,(255,215,0),(LinePos1),(LinePos2),10)#surfa
    ce,colour,start,end,width
600.
601.         pygame.display.update()
```

```
602.
603.     if options[choice] == "play":
604.         game = True
605.     elif options[choice]=="levelSelect": #sets the variables
   for each while loop
606.         levelSelect = True                #depending on what
   was picked in the
607.     elif options[choice] == "highScores":#splash screen menu
608.         highScores = True
609.     elif options[choice] == "intsructions":
610.         instructions = True
611.
612.
613.     while instructions:
614.         win.fill(white)
615.         win.blit(instructionBackgroundImage,(0,0))
616.         draw_text(font1,"Press p to return to main
   menu",black,40,screenWidth//2,800)
617.         win.blit(crystalImage,(700,600))
618.         win.blit(knightRight,(800,600))
619.         win.blit(coin1,(910,620))
620.         win.blit(crouchingLeft[0],(1000,620))
621.         win.blit(ladderImage,(1100,600))
622.         win.blit(ladderImage,(1100,654))
623.         win.blit(ladderImage,(1100,708))
624.         win.blit(ammoCrateImage,(800,700))
625.         win.blit(cakeAttackerImages[0],(945,700))
626.
627.         pygame.display.update()
628.         for event in pygame.event.get():
629.             if event.type == KEYDOWN:
630.                 if event.key == K_p:
631.                     instructions = False
632.                     menu = True
633.         pygame.display.update()
634.
635.
636.
637.     while highScores:
638.         win.fill(bloodred)
639.         fontSize = 30
640.
641.         if oneTimeScoreCalculation:
642.             #extracts scores from file
643.             file = open("highscores/highscores.txt","r")
```

```
644.                lines = file.readlines()
645.                for i in range(len(lines)):
646.                    details = lines[i].split(",")
647.                    scores.append(int(details[0]))
648.                scores.sort(reverse = True)
649.                #creates text to draw from the scores
650.                for i in range(5):
651.                    text = str(i+1)
652.                    points = abs((screenWidth/fontSize) -
    (len(str(scores[i]))+1)) #the number of dots to put between the
    place and the score
653.                    for j in range(int(points)):
654.                        text = text + "."
655.                    text = text + str(scores[i]) # this and few
    lines above make the score thingy e.g. 1.................450
656.                    topFiveScores.append(text)
657.                    text = ""
658.                print(topFiveScores)
659.                oneTimeScoreCalculation = False
660.            #draws the text to draw
661.            for i in range(5):
662.
    draw_text(font1,topFiveScores[i],black,fontSize,screenWidth//2,
    200+125*i)
663.            pygame.display.update()
664.
665.            for event in pygame.event.get():
666.                if event.type == KEYDOWN:
667.                    if event.key == K_p:
668.                        highScores = False
669.                        menu = True
670.            pygame.display.update()
671.
672.
673.        while levelSelect:
674.            pygame.mouse.set_visible(True)# makes the mouse
    visible
675.            pygame.mouse.set_cursor(*pygame.cursors.broken_x)#
    sets cursor to a broken x
676.            mouse = pygame.mouse.get_pos()
677.            win.blit(levelSelectBg,(0,0))
678.            levelLocations =
    [[90,135],[738,135],[1398,135],[405,627],[1050,627]]#top left
    co-ords for each level select level
679.            for i in range(5):
```

```
680.
  win.blit(levelPreviewImageFiller,(levelLocations[i]))
681.         for event in pygame.event.get():
682.             if event.type == KEYDOWN:
683.                 if event.key == K_BACKSPACE:
684.                     levelSelect = False
685.                     menu = True
686.         if pygame.mouse.get_pressed()[0]:
687.             for i in range(len(levelLocations)):
688.                 x = levelLocations[i][0]# if you click on a
  certain level preview
689.                 y = levelLocations[i][1]
690.                 if mouse[0]>x and mouse[0]<(x + 474):
691.                     if mouse[1]>y and mouse[1]<(y+231):
692.                         game_map = levels[i]# level is the
  level you clicked on
693.                         levelSelect = False
694.                         game = True #starts the game
695.                         pygame.mouse.set_visible(False)
696.                         currentLevel = i+1
697.                         print(currentLevel)
698.         pygame.display.update()
699.
700.
701.     while game:
702.         pygame.mouse.set_visible(True)# makes the mouse
  visible
703.         pygame.mouse.set_cursor(*pygame.cursors.broken_x)#
  sets cursor to a broken x
704.         mouse = pygame.mouse.get_pos()
705.         win.fill((11,219,219))
706.         if playerHealth<0:
707.             playerHealth  = 0
708.         drawHealthBar()
709.         drawWeaponCooldown(weaponCooldown)
710.         drawAmmoCounter(playerAmmo)
711.         displayMoneyCounter(playerMoney)
712.         displayKillCounter(enemiesKilled)
713.
  displayCrystalsGathered(crystalsGathered,totalCrystals)
714.
715.
716.         tile_rects = []
717.         allBlocks = []
718.         y =0
```

```
719.         for row in game_map:
720.             x = 0
721.             for tile in row:
722.                 if tile == "o":
723.                     win.blit(outsideCloudImage,((x-1) *
    TILE_SIZEX, y * TILE_SIZEY))
724.                 if tile == "p":
725.                     win.blit(caveCloudImage,((x-1) *
    TILE_SIZEX, y * TILE_SIZEY))
726.                 if tile == "w":
727.                     win.blit(grass, ((x-1) * TILE_SIZEX, y *
    TILE_SIZEY))
728.                 if tile == "s":
729.                     win.blit(dirt, ((x-1) * TILE_SIZEX, y *
    TILE_SIZEY))
730.                 if tile == "i":
731.                     win.blit(grassCobble, ((x-1) * TILE_SIZEX,
    y * TILE_SIZEY))
732.                 if tile == "k":
733.                     win.blit(cobble, ((x-1) * TILE_SIZEX, y *
    TILE_SIZEY))
734.                 if tile == "h":
735.                     win.blit(ladderImage, ((x-1) * TILE_SIZEX,
    y * TILE_SIZEY))
736.                     ladders.append(pygame.Rect((x-1) *
    TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))
737.                 if tile == "c":
738.                     win.blit(crystalImage,((x-1)*TILE_SIZEX,
    y*TILE_SIZEY))
739.                     if oneTimeLevelConstructor:
740.
    crystals.append(pygame.Rect((x-1)*TILE_SIZEX, y*TILE_SIZEY,
    TILE_SIZEX, TILE_SIZEY))
741.                         totalCrystals +=1
742.                 if tile == "a":
743.                     win.blit(ammoCrateImage,((x-1)*TILE_SIZEX,
    y*TILE_SIZEY+4))#the plus four makes images align better
744.                     if oneTimeLevelConstructor:
745.
    ammoCrates.append(pygame.Rect((x-1)*TILE_SIZEX, y*TILE_SIZEY,
    TILE_SIZEX, TILE_SIZEY))
746.                 if tile == "m" :
747.
    win.blit(coinImages[coinCounter//12-1],((x-1)*TILE_SIZEX,
    y*TILE_SIZEY))
```

```
748.                     if oneTimeLevelConstructor:
749.
    coins.append(pygame.Rect((x-1)*TILE_SIZEX, y*TILE_SIZEY,
    TILE_SIZEX, TILE_SIZEY))
750.                 if tile == "1":
751.                     win.blit(lavaSurface, ((x-1) * TILE_SIZEX,
    y * TILE_SIZEY))
752.                     if oneTimeLevelConstructor:
753.                         damageBlocks.append(pygame.Rect((x-1)
    * TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))
    #
754.                 if tile == ".":
755.                     win.blit(lava, ((x-1) * TILE_SIZEX, y *
    TILE_SIZEY))
756.                     if oneTimeLevelConstructor:
757.                         damageBlocks.append(pygame.Rect((x-1)
    * TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))
758.                 if tile =="e":
759.                     if oneTimeLevelConstructor:
760.                         enemies.append(Enemy(x*TILE_SIZEX,
    y*TILE_SIZEY,54,45,400,10,"attacker",2,0,1))
761.                 if tile == "r":
762.                     if oneTimeLevelConstructor:
763.                         enemies.append(Enemy(x*TILE_SIZEX-32,
    y*TILE_SIZEY+15,54,45,400,10,"ranged",2,0,1))
764.                 if tile not in bombCantHitList:#the list of
    blocks for the projectiles to be able to hit(dont want hitting
    air or ladders)
765.                     allBlocks.append([pygame.Rect((x-1) *
    TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY),tile])
766.                 if tile not in cantCollideList :
767.                     tile_rects.append(pygame.Rect((x-1) *
    TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))
768.                 if tile == "o" or tile == "p":
769.                     tile_rects.append(pygame.Rect((x-1) *
    TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))#need two
    blocks as clouds are twice as wide
770.                     tile_rects.append(pygame.Rect((x) *
    TILE_SIZEX, y * TILE_SIZEY, TILE_SIZEX, TILE_SIZEY))
771.                 x += 1
772.             y += 1
773.         oneTimeLevelConstructor = False#so it only adds
    certain blocks to the array once, so it doesnt have an infinitely
    filling up array
774.
```

```
775.         for enemy in enemies:
776.             enemy.draw()
777.             if enemy.category=="ranged":
778.                 enemy.attack()
779.             if enemy.category!="ranged":
780.                 enemy.move()
781.             if enemy.rect.colliderect(player_rect) and
   damageCounter == 0:
782.                 playerHealth -= 25
783.                 damageSound.play()
784.                 if player_rect[0]> enemy.rect[0]:
785.                     damagedSide = 1
786.                 else:
787.                     damagedSide = -1
788.                 isPlayerHit = True
789.             enemy.movementy+=0.4
790.             if enemy.movementy>10:
791.                 enemy.movementy=10
792.         damageCounter +=1
793.         if damageCounter ==15:
794.             damageCounter = 0
795.         if playerHealth<0:
796.             playerHealth = 0
797.
798.
799.
800.
801.
802.
803.         if weaponCooldown<250:
804.             weaponCooldown += 2
805.             if weaponCooldown>250:
806.                 weaponCooldown = 250
807.
808.
809.
810.         if isPlayerAttacking :
811.             shielding = False
812.             if lastFacing == 1:
813.                 sword[0] = player_rect[0]+ 58
814.             else:
815.                 sword[0] = player_rect[0]-(75+sword[3])
816.             sword[1] = player_rect[1]+ 32
817.             pygame.draw.rect(win,red,sword,-1)
818.         else:
```

```
819.             sword[0] = -100
820.             sword[1] = -100
821.
822.         if isPlayerAttacking:
823.             if attackCycle>23:
824.                 attackCycle = 0
825.             attackCycle+=1
826.             if lastFacing == 1:
827.                 playerImage =
  attackAnimationRight[(attackCycle)//6-1]
828.             else :
829.                 playerImage =
  attackAnimationLeft[(attackCycle)//6-1]
830.             attackSound.play()
831.             if attackCycle == 24:
832.                 isPlayerAttacking = False
833.             if lastFacing ==-1:#to make the image line up with
  the coords
834.                 win.blit(playerImage,
  (player_rect.x-(140-64),player_rect.y))
835.             else:
836.                 win.blit(playerImage,
  (player_rect.x,player_rect.y))
837.
838.
839.
840.
841.
842.
843.         for enemy in enemies:
844.             if sword.colliderect(enemy) and
  playerDealDamageCounter == 0:
845.                 enemy.health -= 200
846.                 enemy.rect[0]+=10*lastFacing
847.                 if lastFacing ==1:
848.                     enemy.x += 15
849.                 else:
850.                     enemy.x -=15
851.             for block in damageBlocks:
852.                 if block.colliderect(enemy):
853.                     enemy.health -=10
854.             if enemy.health <=0:
855.                 enemies.pop(enemies.index(enemy))
856.                 enemiesKilled += 1
857.         playerDealDamageCounter +=1
```

```
858.         if playerDealDamageCounter >= 10:
859.             playerDealDamageCounter = 0
860.
861.
862.
863.
864.         if shielding:
865.             isPlayerAttacking = False
866.             if lastFacing ==1:
867.                 shield[0] = player_rect[0] + 65
868.             else:
869.                 shield[0] = player_rect[0] - 6
870.
871.             shield[1] = player_rect[1] + 2
872.
873.             pygame.draw.rect(win,red,shield,-1)
874.             if lastFacing == 1:
875.
   win.blit(shieldImageRight,(shield[0]-12,shield[1]))
876.             else:
877.
   win.blit(shieldImageLeft,(shield[0]-12,shield[1]))
878.         if not shielding:
879.             shield[0] = -200
880.             shield[1] = -200
881.
882.
883.         if pygame.mouse.get_pressed()[0] and playerAmmo>0:
884.             aiming = True
885.         # Drawing aim vectors and circle
886.         # finding values for each projectile vector
887.         if aiming:
888.             aimCircleCenter =
   [(player_rect[0]+player_rect[2]//2),(player_rect[1]+player_rect
   [3]//2)]
889.             aimDeltax =
   mouse[0]-(player_rect[0]+player_rect[2]//2)
890.             aimDeltay =
   mouse[1]-(player_rect[1]+player_rect[3]//2)
891.             radiusAimCircle =
   ((aimDeltax)**2+(aimDeltay)**2)**0.5#radius of circle will be
   from center of player to mouse unless mouse is too far away
   #
892.             aimLineLength = radiusAimCircle
893.             if radiusAimCircle >250:
```

```
894.                 radiusAimCircle = 250
895.             aimLineEndPoint = [(aimCircleCenter[0] +
   aimDeltax*(radiusAimCircle/aimLineLength)),(aimCircleCenter[1]
   + aimDeltay*(radiusAimCircle/aimLineLength))]
896.
   pygame.draw.circle(win,red,(aimCircleCenter[0],aimCircleCenter[
   1]),radiusAimCircle,2)
897.
   pygame.draw.line(win,gold,(aimCircleCenter[0],aimCircleCenter[1
   ]),(aimLineEndPoint[0],aimLineEndPoint[1]),3)
898.
899.
900.
   pygame.draw.circle(win,gold,(aimLineEndPoint[0],aimLineEndPoint
   [1]),5)#next 4 shapes drawm are for aesthetics
901.
   pygame.draw.circle(win,green,(aimLineEndPoint[0],aimCircleCente
   r[1]),5)
902.
   pygame.draw.line(win,green,(aimCircleCenter[0],aimCircleCenter[
   1]),(aimLineEndPoint[0],aimCircleCenter[1]),3)
903.
   pygame.draw.line(win,green,(aimLineEndPoint[0],aimCircleCenter[
   1]),(aimLineEndPoint[0],aimLineEndPoint[1]),3)
904.             if  not pygame.mouse.get_pressed()[0]:
905.                 aiming = False
906.                 aimed = True
907.                 aimVectorDeltax = aimLineEndPoint[0] -
   aimCircleCenter[0]
908.                 aimVectorDeltay = aimLineEndPoint[1] -
   aimCircleCenter[1]
909.
910.         # firing projecti;es
911.         if  len(bombs)<4 and shotsFiredCounter == 0 and
   playerAmmo>0 and aimed :
912.
   bombs.append(Bomb(15,player_rect,299,aimVectorDeltax,aimVectorD
   eltay,bombImage))
913.             playerAmmo-=1
914.             shotsFiredCounter += 1
915.             aimed = False
916.
917.         shotsFiredCounter+=1
918.         if shotsFiredCounter >7:
919.             shotsFiredCounter = 0
```

```
920.
921.
922.
923.            # making projectiles do damage, deleting projectiles
  when they hit things, making projectiles break blocks
924.            for bomb in bombs:
925.                bomb.move()
926.                bomb.draw()
927.                isCollidedTile,isCollidedEnemy, tileCollided,
  enemyCollided = bomb.collisionCheck(allBlocks)
928.                if isCollidedTile:
929.                    if not(tileCollided[1] in cantCollideList):
930.
  game_map[(tileCollided[0][1]//TILE_SIZEY)][(tileCollided[0][0]/
  /TILE_SIZEX +1)] = " "
931.                        explosionSound.play()
932.                # must delete in the same place otherwise if bullet
  hits tile and enemy, will try to delete bullet twice otherwise
933.                #which is not possible
934.                if isCollidedEnemy:
935.                    enemyCollided.health -= bomb.damage
936.                    explosionSound.play()
937.                if isCollidedEnemy or isCollidedTile:
938.                    bombs.pop(bombs.index(bomb))
939.
940.
941.            for orb in orbs:
942.                orb.move(player_rect)
943.                orb.draw()
944.                if orb.x>1920 or orb.x<0 or orb.y>1080 or orb.y<0:
945.                    orbs.pop(orbs.index(orb))
946.                if player_rect.colliderect(orb.rect):
947.                    playerHealth -= orb.damage
948.                    damageSound.play()
949.                    orbs.pop(orbs.index(orb))
950.                if shield.colliderect(orb.rect):
951.                    shieldSound.play()
952.                    try:
953.                        orbs.pop(orbs.index(orb))
954.                    except:
955.                        pass
956.
957.
958.            #making the player bounce back/take knockback when hit
959.            if isPlayerHit:
```

```
960.             player_y_vel = -9
961.             if playerKnockbackCounter <2:
962.                 player_rect,collisions =
  move(player_rect,[(20*damagedSide),0],tile_rects)
963.                 playerKnockbackCounter = 0
964.                 isPlayerHit = False
965.             playerKnockbackCounter +=1
966.
967.         player_movement = [0,0]
968.
969.         # crouching code
970.         if crouching:
971.             player_rect[3]=50
972.             if lastFacing == 1:
973.                 playerImage = crouchRight1
974.             else:
975.                 playerImage = crouchLeft1
976.             if crouchCycle>23:
977.                 crouchCycle = 0
978.             crouchCycle +=1
979.
980.             if moving_right:
981.                 player_movement[0] = 2
982.                 playerImage =
  crouchingRight[(crouchCycle)//12-1]
983.                 lastFacing = 1
984.             if moving_left:
985.                 player_movement[0] = -2
986.                 playerImage =
  crouchingLeft[(crouchCycle)//12-1]
987.                 lastFacing = -1
988.
989.
990.         #moving right
991.         if moving_right and not crouching and not
  isPlayerAttacking :
992.             if walkCycle>26:
993.                 walkCycle = 0
994.             walkCycle +=1
995.             player_movement[0]=5
996.             if isJump ==False:
997.                 playerImage = runningRight[(walkCycle)//9-1]
998.             else:
999.                 playerImage = runningRight[1]
1000.            lastFacing = 1
```

```
1001.
1002.         # moving left
1003.         if moving_left and not crouching and not
  isPlayerAttacking:
1004.             if walkCycle>26:
1005.                 walkCycle = 0
1006.             walkCycle+=1
1007.             player_movement[0]-=5
1008.             if isJump == False :
1009.                 playerImage = runningLeft[(walkCycle)//9-1]
1010.             else:
1011.                 playerImage = runningLeft[1]
1012.             lastFacing = -1
1013.
1014.
1015.         #Climbing,rest of the code is down by the keybind
  section so I can do things when "w" is pressed
1016.         if climbing:
1017.             player_y_vel = 0
1018.             shielding = False
1019.             if canClimb == False:
1020.                 climbing = False
1021.             else:
1022.                 if climbingCycle>23:#if the player can climb,
  then they climb
1023.                     climbingCycle = 0
1024.                 climbingCycle +=1
1025.                 playerImage =
  climbingImages[(climbingCycle)//12-1]
1026.                 player_movement[1] -= 5
1027.
1028.
1029.
1030.         # standing still
1031.         if (not moving_left) and (not moving_right) and (not
  climbing) and (not crouching) and (not isPlayerAttacking) :
1032.             if lastFacing==-1:
1033.                 playerImage = knightLeft
1034.             if lastFacing == 1:
1035.                 playerImage = knightRight
1036.
1037.
1038.         player_movement[1] += player_y_vel
1039.         if not climbing:
1040.             player_y_vel += 0.4
```

```
        if player_y_vel > 10:
            player_y_vel = 10
        # collisions,falling down, and moving the player
        # also refreshing player image
        player_rect, collisions = move(player_rect,
  player_movement, tile_rects)
        if collisions["bottom"]:
            player_y_vel = 0
            air_timer = 0
            isJump = False
        else:
            air_timer += 1

        if playerHealth>0 and not isPlayerAttacking:
                win.blit(playerImage,
  (player_rect.x,player_rect.y))



        for event in pygame.event.get(): # event loop
            if event.type == QUIT: # check for window quit
                pygame.quit() # stop pygame
                sys.exit() # stop script
            if event.type == KEYDOWN:
                if event.key == K_w:
                    movingRight = False
                    movingLeft = False
                    climbing = True
                    canClimb = False
                    for ladder in ladders:
                        if player_rect.colliderect(ladder):
                            canClimb = True
                            if oneTime:
                                player_rect[0]=ladder.x
                                player_rect[1]=ladder.y
                                oneTime = False
                    oneTime = True
                # key binds
                if event.key == K_d:
                    moving_right = True
                if event.key == K_a:
                    moving_left = True
                if event.key == K_s:
                    crouching = True
```

```
1084.                 if event.key == K_SPACE:
1085.                     isJump = True
1086.                     if air_timer < 6:
1087.                         player_y_vel = -9
1088.                         jumpSound.play()
1089.                 if event.key == K_RETURN and weaponCooldown ==
   250:
1090.                     weaponCooldown = 0
1091.                     isPlayerAttacking = True
1092.                 if event.key == K_p:
1093.                     game = False
1094.                 if event.key == K_LSHIFT:
1095.                     shielding = True
1096.
1097.
1098.             if event.type == KEYUP:
1099.                 if event.key == K_w:
1100.                     climbing = False
1101.                 if event.key == K_d:
1102.                     moving_right = False
1103.                 if event.key == K_a:
1104.                     moving_left = False
1105.                 if event.key == K_s and canStand():
1106.                     crouching = False
1107.                     isCanStand = True
1108.                     player_rect[3] = 64
1109.                 # dont need for keyup on enter, as will set
   isPlayerAttacking to false when the animation is finsihed
1110.
1111.                 if event.key == K_LSHIFT:
1112.                     shielding = False
1113.
1114.
1115. #taking damage from members of the damage block list
1116.             for block in damageBlocks:
1117.                 if player_rect.colliderect(block) and
   healthTimer == 0:
1118.                     damageSound.play()
1119.                     playerHealth -=10
1120.                     healthTimer +=1
1121.                     if playerHealth<0:
1122.                         playerHealth = 0
1123. #making sure health timer doesnt go down too fast
1124.             if healthTimer>0:
1125.                 healthTimer +=1
```

```
1126.             if healthTimer>5:
1127.                 healthTimer = 0
1128.
1129. #displaying coin images, removing coins when touching them
1130.         if coinCounter>23:
1131.             coinCounter = 0
1132.         coinCounter+=1
1133.         for coin in coins:
1134.             if player_rect.colliderect(coin):
1135.                 coinSound.play()
1136.                 playerMoney += coinValue
1137.                 coins.pop(coins.index(coin))
1138.
    game_map[(coin[1]//TILE_SIZEY)][(coin[0]//TILE_SIZEX +1)] = "
    "#removes coin image from the game map so wont be redrawn even if
    already taken by player
1139.
1140.
1141. #colliding with crystals and removing them from the map
1142.         for crystal in crystals:
1143.             if player_rect.colliderect(crystal):
1144.                 crystalsGathered +=1
1145.                 crystals.pop(crystals.index(crystal))
1146.
    game_map[(crystal[1]//TILE_SIZEY)][(crystal[0]//TILE_SIZEX +1)]
    = " "
1147.
1148. #collecting ammo from crates
1149.         for ammoCrate in ammoCrates :
1150.             if player_rect.colliderect(ammoCrate) and
    playerAmmo<playerMaxAmmo:
1151.                 playerAmmo = playerMaxAmmo
1152.                 ammoCrates.pop(ammoCrates.index(ammoCrate))
1153.
    game_map[(ammoCrate[1]//TILE_SIZEY)][(ammoCrate[0]//TILE_SIZEX
    +1)] = " "
1154.
1155.         if playerHealth <=0:
1156.
    win.blit(knightDeadImage,(player_rect[0],player_rect[1]+14))
1157.             pygame.display.update()
1158.             pygame.time.wait(1000)
1159.             height = 0
1160.             while height<screenHeight:
```

```
1161.
   pygame.draw.rect(win,bloodred,(0,0,screenWidth,round(height)))
1162.                 pygame.display.update()
1163.                 height+=1.4
1164.             playerAmmo = 5
1165.             menu,game,totalScore = gameOver(totalScore)
1166.             oneTimeScoreCalculation = True
1167.             file = open("highscores/highscores.txt","a")
1168.             file.write("\n")
1169.             file.write(str(totalScore)+",")
1170.             file.close()
1171.             totalScore = 0
1172.
1173.
1174.         if crystalsGathered ==  totalCrystals :
1175.             oneTimeScoreCalculation = True
1176.             totalScore = endLevelScores(totalScore)
1177.             if currentLevel<5:
1178.                 win.fill((146,219,1))
1179.                 draw_text(font1,("Coins Collected:  " +
   str(playerMoney)),black,30,970,500)
1180.                 pygame.display.update()
1181.                 pygame.time.wait(1000)
1182.                 draw_text(font1,("Enemies Defeated:  " +
   str(enemiesKilled)),black,30,970,545)
1183.                 pygame.display.update()
1184.                 pygame.time.wait(1000)
1185.                 draw_text(font1,("Total Score:  " +
   str(playerMoney*50 + enemiesKilled*100)),black,30,970,590)
1186.                 pygame.display.update()
1187.                 pygame.time.wait(1000)
1188.
   draw_text(font1,("level"+(str(currentLevel+1))),red,40,(970),(6
   50))
1189.                 pygame.display.update()
1190.                 pygame.time.wait(2000)
1191.             currentLevel +=1
1192.             try:
1193.                 game_map = levels[currentLevel-1]
1194.             except:
1195.                 pass
1196.             enemies = []
1197.             crystals=[]
1198.             coins=[]
1199.
```

```
            oneTimeLevelConstructor = True
            moving_right = False
            moving_left = False
            climbing = False

            playerHealth = 250
            player_rect.x = 10
            player_rect.y = 10
            crystalsGathered = 0
            totalCrystals = 0
            playerMoney = 0
            enemiesKilled = 0
            playerAmmo = playerMaxAmmo

        if (currentLevel == 6):
            menu, game, totalScore = victory()




        pygame.display.update() # update display
        clock.tick(60) # maintain 60 fps


    #below is if you exit the game loop, it resets all the values so you should be able to replay from the start from the menu
crystalsGathered,oneTimeLevelConstructor,moving_right,moving_left,climbing,crystals,coins,totalCrystals,player_rect.x,player_rect.y,playerHealth,playerMoney = resetValues()
    currentLevel = 1
    enemies = []
    level1 = importLevel(1)
    level2 = importLevel(2) #resets all the levels so they can be replayed
    level3 = importLevel(3)
    level4 = importLevel(4)
    level5 = importLevel(5)
    game_map = level1

    menu = True

```