# A Level Computer Science

OCR H446 - 03

Unit 3 - Programming Project

**Candidate Name:** Tasfia Tasnim Saiyeda

**Candidate Number:** 9312

**Centre Name:** King Edward VI Handsworth School for Girls

**Centre Number:** 20155

**Project Title:** Cat Fight

# Contents

[Analysis of the problem 2](#_nczmthvfw20d)

[Problem Identification 2](#_i4e3ztxvtjnr)

[User survey 2](#_fds02yun9ovb)

[Stakeholders 4](#_c0tdmcy14ngi)

[Research the problem 5](#_cf1jdj454w0j)

[Street Fighter (1987) 5](#_isuv7i23mfgo)

[Super Smash Bros. 6](#_rqrwn0s9x0w5)

[Marvel Super Heroes vs. Street Fighter 8](#_8vxzmaa00wye)

[Naruto: Ultimate Ninja Storm 8](#_3h2d2niqg2ma)

[Specify the proposed solution 9](#_j6gsn2fs9bn)

[User survey 9](#_ppl2x44ibpxp)

[Design of the program 10](#_grnmvo2h5f14)

[Functionality of the program 11](#_yg891poqjrv3)

[Hardware and software required for the program 12](#_trnru59zb5hm)

[Success criteria for the program 13](#_sw9sryozg5z)

[Limitations 14](#_wkitekrqz3j6)

[Design of the solution 15](#_juv8d4f23e6c)

[Decompose the solution 15](#_mjucyh78vlsr)

[Top down design diagram 15](#_sqww2464ngcm)

[Battle 16](#_x3wn2kmfgjbs)

[Starting page 44](#_z7mtngkjw6zv)

[Character Selection 45](#_y4afviuhwj3d)

[Describe the solution 46](#_7hsq7jjj03k)

[Design objectives 46](#_w3ot0wwtzc18)

[Aesthetic considerations 47](#_q8pct5uuwl71)

[Input considerations 47](#_kn8se1gpkma3)

[Processing considerations 48](#_qzgdmrquuox8)

[Output considerations 48](#_hb767iitgzpo)

[Interface design 48](#_bzm9g7silqb4)

[Describe the approach to testing 50](#_8qegiw8lh5o1)

[Developing the solution 55](#_qgsgs0rqr2du)

[Iterative development process 55](#_ig8e5crdtjri)

[Create the starting page 55](#_vl8autyvcxut)

[Create the character selection 56](#_684xhvwf8j4p)

[Write the battle code 58](#_om7y4chyg28l)

[Create the after battle screen 68](#_k6i80z7cvag2)

[Testing to inform development 75](#_ispli94w3ey)

[Evaluation 85](#_5crvyuqcx96e)

[Testing to inform evaluation 85](#_yzjhu82ob8ob)

[User questionnaire 85](#_rb6xaldkotcg)

[Success of the solution 86](#_nncw6sat6odz)

[Describe the final product 92](#_2obx8zdlbozv)

[Maintenance and development 99](#_cza7bplhpp1h)

[Good points 99](#_2y9ao19upibi)

[Bad points 99](#_5jrzjoghm1gz)

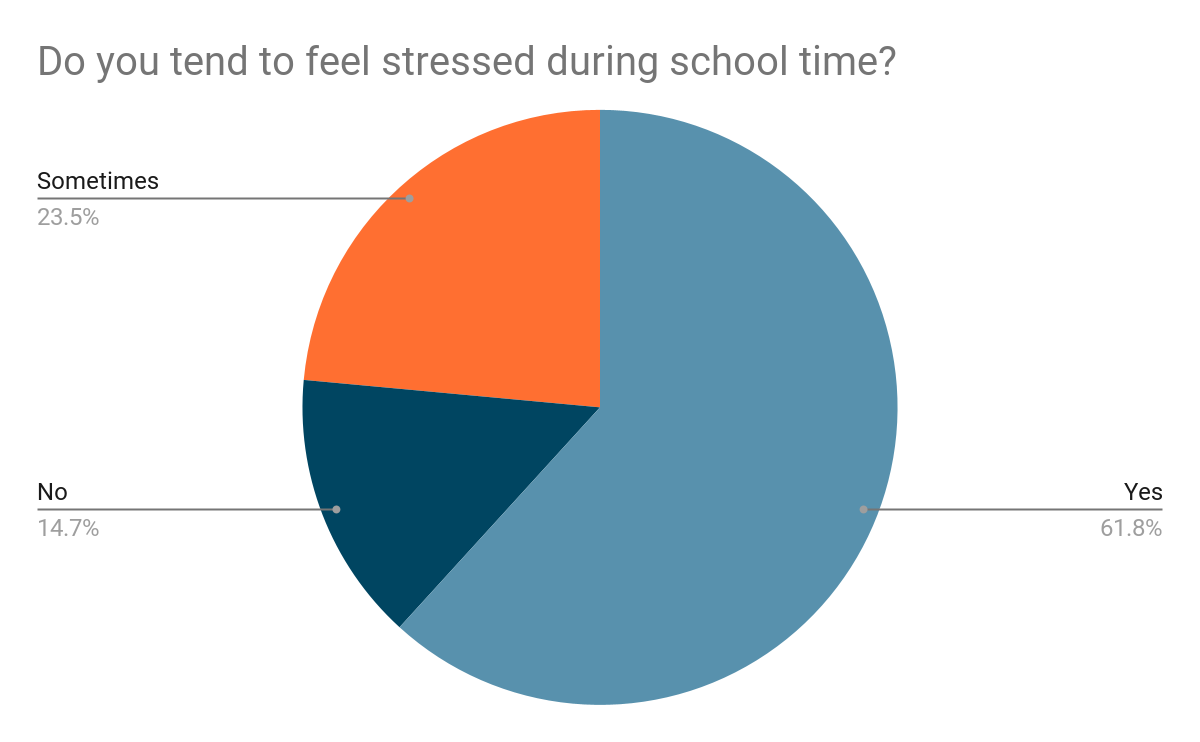
[Limitations and how they may be resolved 99](#_i0bio61jfvht)

[Bibliography 101](#_pr4iy7wqg7ww)

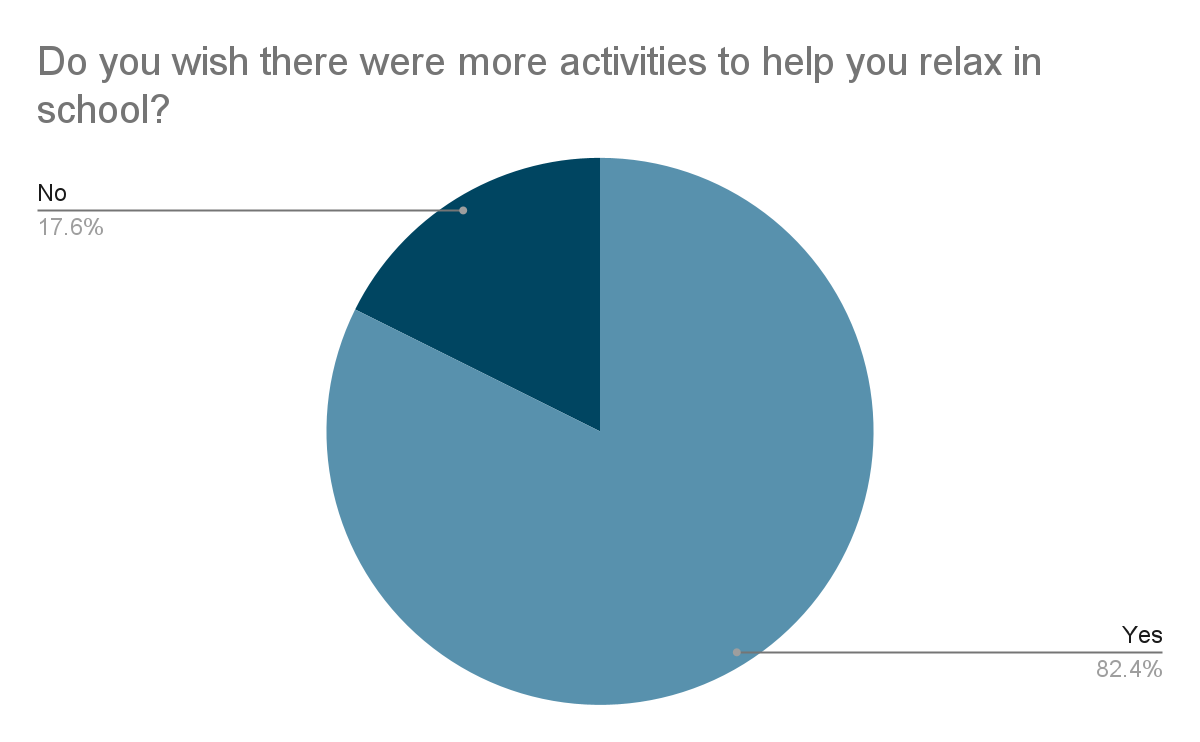
[Appendix 102](#_cxrw0m2z3iwm)

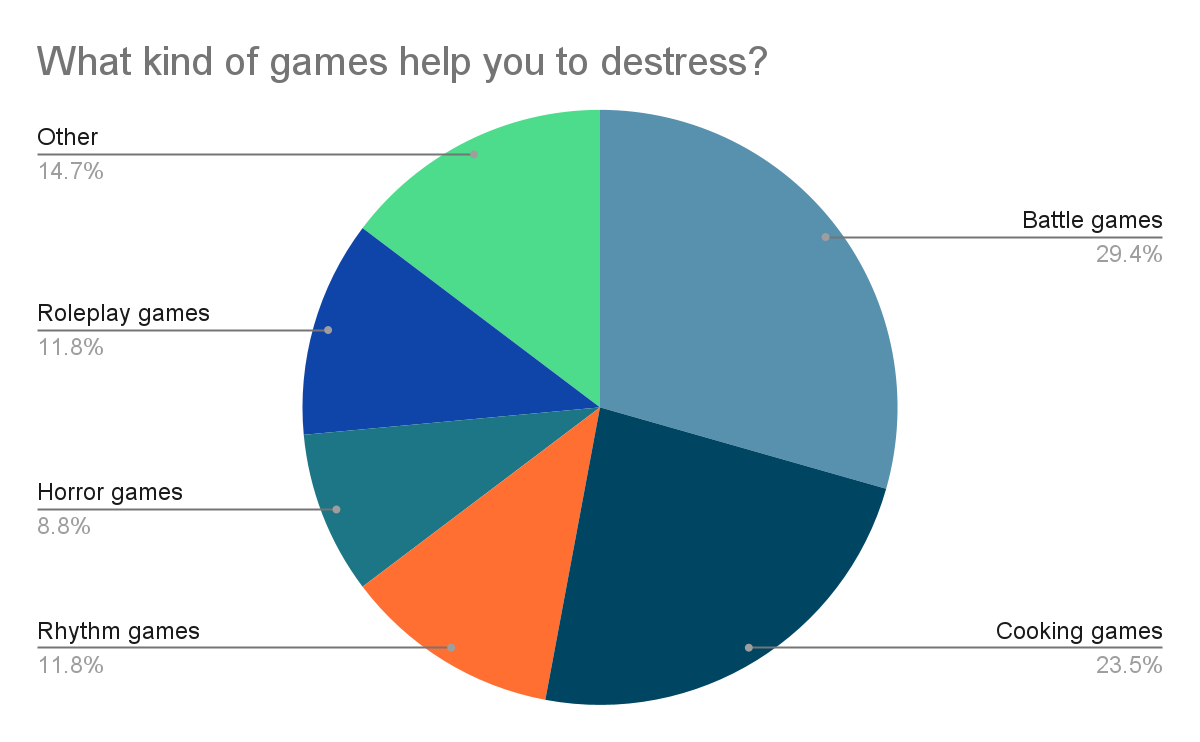
### Analysis of the problem

#### Problem Identification

Students studying for GCSE and A level exams are often working hard in content heavy lessons at school in order to prepare for their exams with only 15-30 minute breaks between long double lessons may want something to help them relax and have fun in their breaks, especially if there aren’t any suitable clubs or activities to attend. I asked students currently in Year 10 to 13 some questions about stress and what they do to relax and these were the results. 

##### User surveyPoints scored





From this data, it can be seen that GCSE and A Level students do desire a form of relaxation between their lessons, and playing games is a popular form of relaxation. As battle games are a favoured choice of game to help these students destress, I have decided to make a 2D fighter game to help solve this problem.

This problem can be solved with computational methods such as abstraction. Unneeded features of a potential 2D fighter game could be abstracted such as complex character design or world design. Unnecessary information can be abstracted to solve the problem such as detailed graphics, and information can be abstracted from the users about for example how the characters move other than because of the user using the arrow keys.

This problem can be solved with computational methods such as decomposition. A top down design diagram can be used to split the large problem into smaller sub-problems, and functions can be used within the code to solve these smaller sub-problems. Components of the problem can be decomposed into smaller subproblems and can determine the components of the 2D fighter game and be made into different procedures and functions, which can be reused.

Preconditions for playing the game can be determined such as the need for a keyboard with WASD and arrow keys as well as any other necessary keys that must be inputted. Decisions must be made within the solution such as scoring priorities when players attack simultaneously, or how to display the game and how this would affect the player. Sub-problems decomposed into the same level may be able to be worked on simultaneously such as scoring and health priorities.

Inputs may include characters the players may select, as well as the keyboard inputs the players use to move or attack, and buttons on screen pressed in order to pause or restart the game, for example. Outputs may include displaying the selected characters on screen, and making the characters move or attack on screen when a key is pressed on the keyboard, and making the game stop and start again on screen when prompted by the button pressed. Calculations to be made could include decreasing an opponent’s health when attacked, and the distance the opponent is away from the player so that if the player is too far away from them, they will be unable to attack. These all ensure that the 2D fighter game is easy to play and includes the main features of fighting games.

Specific hardware such as keyboards would need to be used to interact with the solution, as well as a suitable operating system for the software to run on. The game will be run offline on the operating system, and would need to be interacted with by the user with the use of the keyboard keys and the mouse. This requires a computational approach as the user would be interacting with a computer.

Using decomposition to create a top-down design diagram to specify what functions and procedures need to be used in the solution means that it can be represented as an algorithm. The top-down design diagram will also help to see what parts of the problem may take more time than others, therefore can help to make sure the problem can be solved within a reasonable timescale. As I will be using Python to code the solution, the problem will also be solved at low/zero cost, as all school computers already have Python installed. All of these features ensure that the problem is computable.

#### Stakeholders

GCSE and A Level students who are not allowed to use their phones in school can come to a computer room at break and lunch and play this game with each other or against a computer to relieve tension and cool off without using their phones and breaking the school rules. This solution will give them an activity to do during school that will help them destress and have fun, especially as, as shown in the survey, students desire more activities in school that will help them relax. As games are the most preferred way to relax and battle games are the most popular games to help students destress in this sample of students, a 2D fighter game is appropriate to their needs. By researching similar fighting games, I can find out what sort of features are suitable to these students and which features will be able to be coded within a reasonable timescale.

Teachers and parents can supervise gameplay as well as benefit from calmer, more relaxed students and children that can focus better on their work after cooling off. This solution will not only help students to feel more relaxed in the classroom, but teachers will be able to have a more focussed class as the students have been given an opportunity to destress and get their minds off things during break or lunch, which is especially beneficial when it comes to looming exams where it is imperative that students are focussed and ready to learn. Parents will also be able to help their children to revise better at home after they have been given the opportunity to destress and not constantly have the thought of their exams on their mind.

Game developers may also benefit from more people being interested in gaming, and in particular fighting games. This may increase demand for these games, making them more popular, especially as a way to destress, therefore game developers of such games may benefit from more people playing their games.

#### Research the problem

Similar existing solutions are described below:

##### Street Fighter (1987)

Players compete in one-one matches against computer-controlled opponents or in a single match against another player. Each match has three rounds in which the player must knock out the opponent within 30 seconds. If a match ends before a fighter is knocked out, the fighter with the greater amount of energy left is the round’s winner.

To defeat the opponent, the player must win two rounds and then will proceed to the next battle. If the third round ends in a tie, the computer will win by default, or both players will lose. Against a computer, the losing player can continue against the same opponent. A second player can interrupt a single-player match and challenge the first player to a new match.

Attacks consist of six attack buttons, three punch buttons, and three kick buttons of different speeds and strengths: light, medium and heavy. A joystick is used to move left or right, and to jump, crouch and block. The player can perform a variety of attacks from standing, jumping or crouching positions with different combinations of attack buttons along with the joystick. There are three special techniques that require a specific series of inputs.

Single-player mode consists of a series of battles against two opponents in each country for a total of five countries.

The player takes control of a single character named Ryu, and the second player takes control of Ryu’s rival Ken. Both characters have the same basic moves and special techniques.

I like the aspect of being able to play against both a computer and a second player. This would allow students to play not only with their friends, but also give students the opportunity to destress even if they do not have someone to play with. I also like how there are a number of rounds and a time limit. This would make the gameplay more interesting and exciting for players when playing against an opponent, as they can defeat their opponent in different ways, such as having more health or energy than them when the time runs out. I like how there are different types of attacks. This diversifies the gameplay, which in turn will make the game more fun for players as they can perform a selection of attacks.

I would like to have a more diverse character selection so that players are able to have more choice, however I could use the aspect of having the same basic moves and special techniques between characters, so that the gameplay isn’t confusing, especially for students who aren’t experienced in battle games but still want the opportunity to destress with their friends.

##### Super Smash Bros.

The aim of players is to launch their opponents off the stage and out of bounds. Characters have a damage total which increases as they take damage, which is represented by a percentage value up to 999%. The higher the percentage, the stronger knockback the player endures from enemy attacks. To knock out an opponent, the player must knock the character out of the stage’s boundaries. When knocked off stage, the character can attempt to recover by using jumping moves and abilities to return. Some characters vary in weight, with lighter characters easier to launch.

One button is used for standard attacks, and another button is used for special attacks. Different types of moves can be performed by holding the directional controls up, down, to the side, or in a neutral position while pressing the attack or special button. Each character has four types of ground attacks, mid-air attacks and special attacks, as well as a chargeable “Smash Attack” on the ground which is more powerful than other attacks. When characters are hit by attacks, a hitstun is received which temporarily stops the player from being able to perform any attacks, which allows combos to be performed upon the stunned character. A shield button is used to allow players to put up a defensive shield which weakens with repeated use and if broken, renders the player unable to move. The three basic actions (attacking, grabbing and shielding) use a rock-paper-scissors analogy: attacking beats grabbing, grabbing beats shielding, and shielding beats attacking. An action called edge-guarding can be performed when a player knocks another player off the main platform, and the player that has been knocked off will try to recover by trying to jump back onto the stage and avoiding the other players’ edge-guarding.

Players use battle items to assist them in battle, which they can adjust before matches which includes battering items to hit opponents, throwing items and shooting items. Recovery items can be used to reduce the character’s damage percentage by varying amounts. Some special items can be used to release another character onto the battlefield to temporarily assist the player. A “Smash Ball” allows the character to perform a character-specific attack when broken called a “Final Smash”.

There are two different modes: Time and Stock. Time mode uses a point-based system, where players earn points for knocking out their opponents and lose points for being knocked out or falling off the stage themselves. At the end of the time limit set, the player with the most points wins the match. Stock mode/Survival mode uses a life-based system where players are given a set number of lives (stock), and a life is lost whenever the character is knocked out, and they are eliminated when all lives are lost. The winner is the last player standing after all other players have been eliminated, or if a time-limit has been applied, the winner is the player with the most lives left once time runs out. If there is a tie, a Sudden Death match takes place where each player is given a starting damage percentile of 300% which makes it easier for them to be launched off stage, and the last player standing will be the winner. This is repeated if the match ends in another tie.

There are multiple playable characters. Players are able to customise existing characters with altered movesets and abilities, as well as making their own avatar – a Mii – that can be given three different fighting styles.

I like that there are a selection of different characters to choose from with different abilities and attacks. As mentioned before, this allows players to have a more diversified gameplay, and gives them the ability to choose how they would like to play, which would help them to relax as they can play in a way that is preferred to them. I like how different buttons are used for different attacks as well as the ability to charge more powerful attacks. This allows for a more engaging gameplay where players can attack their opponents with a range of different attacks with different effects on the opponent, making it more fun for students who want to beat their friends.

I would like to have a more battle-like gameplay rather than knocking opponents offstage, as specified in the survey. I like the idea of having a combination of a life-based and time limit system. This would allow for players to be defeated in different ways as mentioned earlier, by either losing all of their health or by having the least amount of health when the time runs out. This makes it more fun for players as they can try to defeat their opponent in different ways.

##### Marvel Super Heroes vs. Street Fighter

This game uses a one-on-one tag team format. The player chooses a team of two fighters each with their own health bar. The first selected character is controlled by the player and the second character remains off-screen as a support character.

Combinations of joystick movements and button presses are used to perform various moves to deplete the opponent’s health. The first player to completely drain the opponent’s health is the winner. If the time runs out, the player with the most health wins.

The player can summon their support character to perform a special move without changing the currently-controlled character. This gives new possibilities for combos during battle.

I like the combination of having two different ways to win: by defeating your opponent, or having the most health after a time limit has been reached, as mentioned earlier.

I don't think I would implement support characters as it may be too time consuming but I could potentially add in combination attacks with different key inputs for different characters, allowing for more choice and a more enticing battle where players can perform a range of different attacks against their opponent.

##### Naruto: Ultimate Ninja Storm

This game has a transformation mode that can be activated after a player loses a certain amount of health. The health requirement for each character is based on how powerful the transformation is. When activated, the character gains new abilities, speed and stronger attacks.

Players can use the d-pad to use preset items during battle that can damage the opponents or apply different buffs to the player, or to lower the opponent’s defence.

The player can perform a variety of attacks with different combinations of attack buttons. There are special techniques that require a specific combination of inputs.

Players can customise their special attacks and select two support characters to use in a match. The support characters can be called upon during battle to perform a skill, and are then put on a cooldown until they can be used again.

An energy meter is used to charge special attacks. Players are unable to use a special attack if they have insufficient energy. Energy can be charged by holding a button during battle.

When both players use their special attack at the same time, both players have to press a button shown on screen as fast as possible in order to knock away their opponent.

If the player performs this ultimate attack, both players either input button commands, mash a certain button or spin the analogue stick the fastest within the time limit. If the attacking player wins, the opponents will be hit and about a third of their health bar is taken away. If the defending player wins, they escape without major damage.

There are 25 playable characters that can also be used as a support character.

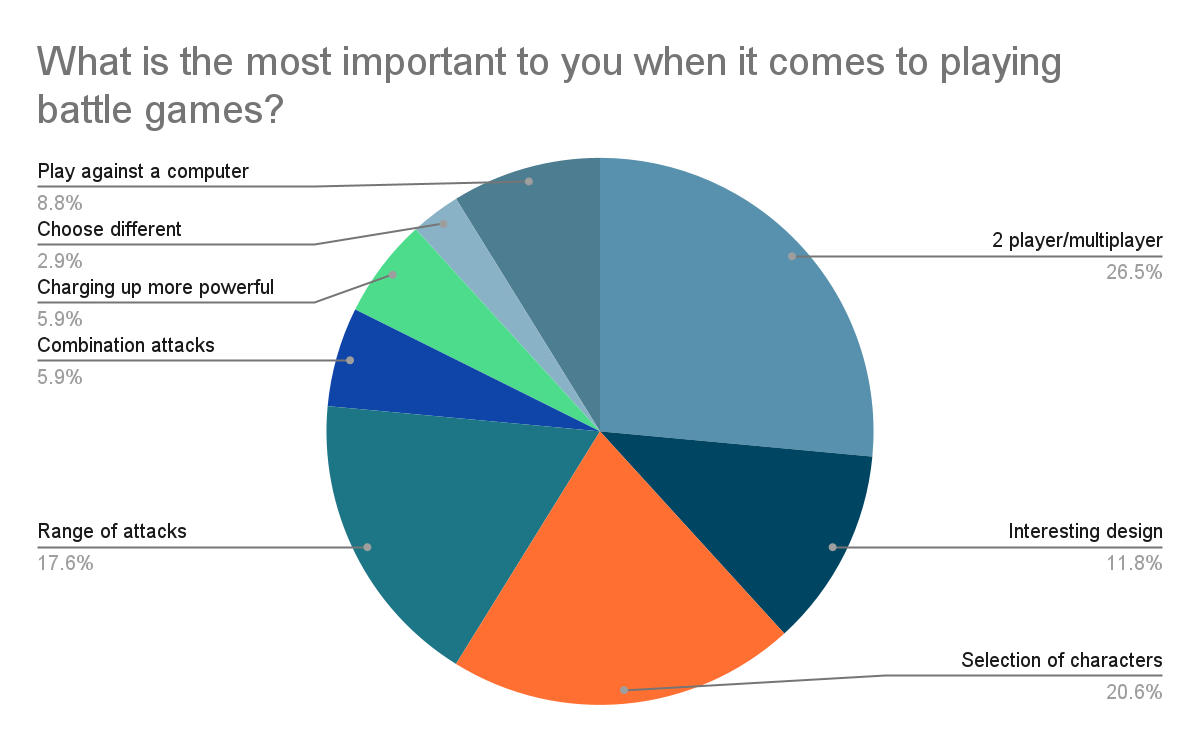
I like the wide variety of characters to choose from and would like to implement a character selection screen, as this would give players more choice in how they would like to play, and make it so that both players do not get mixed up with each other on screen. I like the ability to be able to do special attacks with certain inputs. I think this would make the game more fun as players can try different combinations of attacks against their opponents and see which attacks are the most effective. I also like the energy metre system and having to charge up energy to be able to use special attacks, which would make the battle more engaging and make it last longer, as well as help students to strategise their attacks and how they might try to inflict more damage onto their opponent, so that the game isn’t too relaxing and the students are still ready to go back to lesson with sharp minds after break or lunch.

I don’t think I would implement a special mode for when players lose a certain amount of health, or the use of support characters, as these might be quite time consuming or could clutter the game screen as I am creating a 2D fighter game rather than a 3D fighter game like the one described, potentially making the game more stressful to play instead. The clash mode may be a good idea for when players attack at the same time, and processing considerations must be made such as cooldown times for attacks for example so that players do not constantly attack at the same time. This would also make the battle last longer and encourage students to strategise their approach to defeating their opponent.

#### Specify the proposed solution

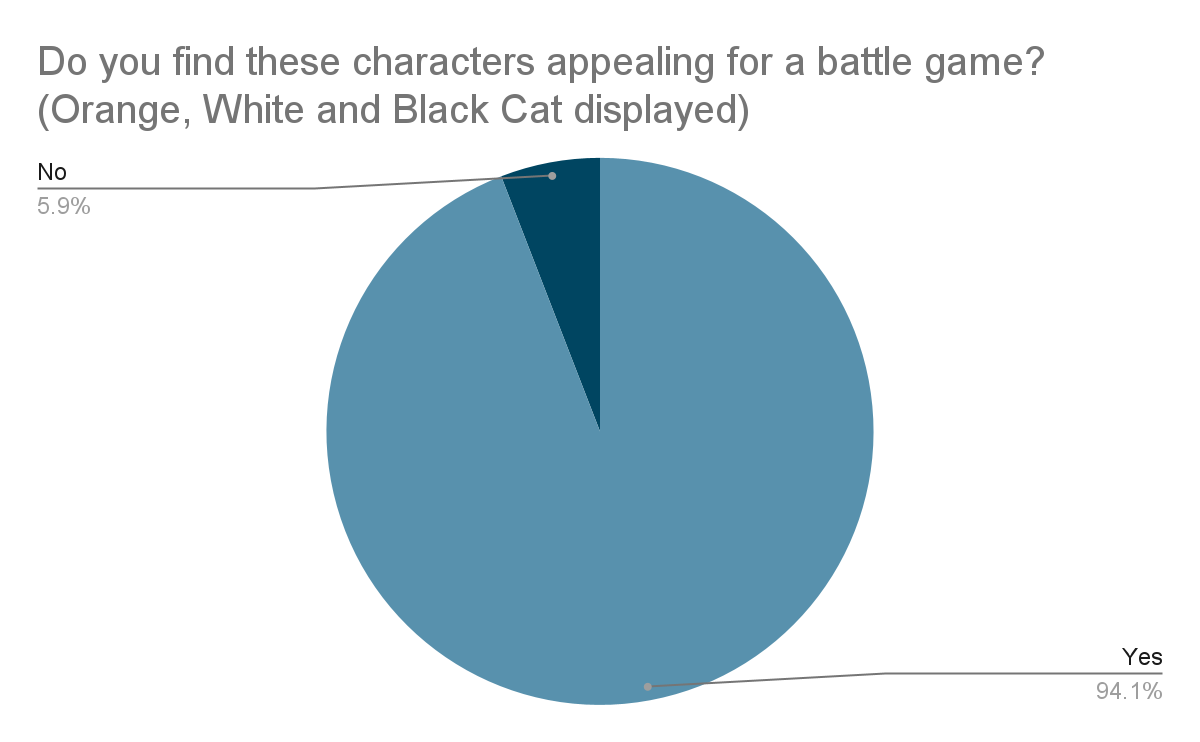
After surveying potential users (GCSE and A Level students) on their preferred battle game requirements, this is what I found:

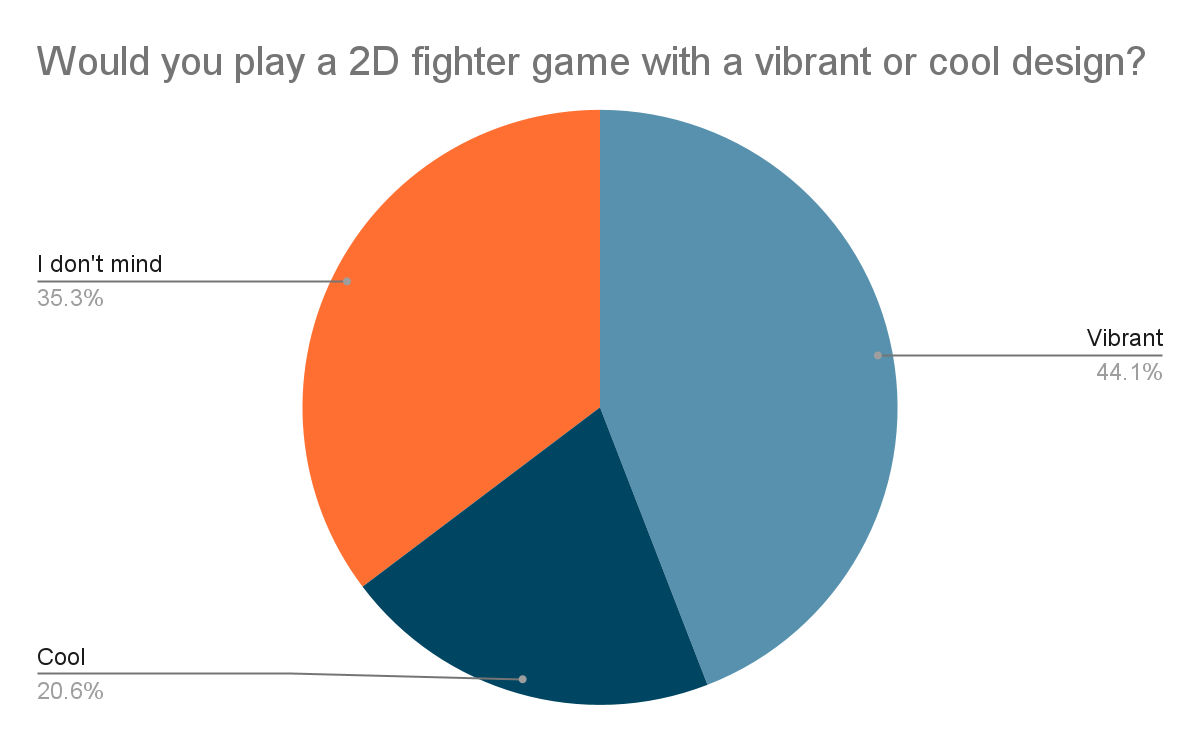
##### User survey











As seen from this user survey, a preferred 2D battle game would be one that has the ability to play against a computer or against friends, has a selection of characters and a range of attacks, with a vibrant design. Using the results of this survey, I shall list out the requirements for my solution below.

##### Design of the program:

| **Requirement** | **Explanation** |
| --- | --- |
| Starting page with start button | An engaging and enticing starting page invites the player into playing the game |
| A ‘How to play’ button on the starting page | Clear instructions are needed to show the controls for both player 1 and player 2 so that the players know how to play the game |
| A settings menu on the starting page | The player will have the ability to change the amount of rounds they would like to play in a game, or to adjust the time limit within a round: 60 seconds, 90 seconds, 120 seconds or no time limit. This gives players the choice that best suits them on how they want to play that will help them to relax the most |
| During battle, a pause button, which, when clicked with the mouse, brings up three buttons: ‘Retry’, ‘Back to character selection’ and ‘Back to starting page’ | If any player needs to stop the game or would like to try again or stop playing, which is helpful if players need to leave and would like to continue their battle later |
| After battle, a button to retry, go back to character selection, and go back to starting page | Reduces the amount work the user will need to put in if they would like to start another battle, change the settings or stop playing, which also helps to save time especially at break time or lunchtime where time is limited |
| Character selection page with the ability to choose between two different special attacks | Gives the players more options to enhance their playing experience and play in a way that is more fun for them |
| Energy meter for each player displayed during battle | Players can see how much energy they have and can use for special or ultimate attacks to try and defeat their opponent quicker |
| Health bar for each character displayed during battle | Players can see how much health they or their opponent have left and can potentially strategise their next moves in this way |

##### 

##### Functionality of the program:

| **Requirement** | **Explanation** |
| --- | --- |
| Movement using WASD for player 1 and arrow keys for player 2 | The characters can move left or right, jump, or guard to evade attacks |
| Three different types of attacks: normal attack, special attack and ultimate attack with three different keyboard inputs | The characters can have their own special and ultimate attacks which creates a unique experience for the player and allows more diversity when it comes to character selection and avoids confusion during battle |
| Buttons on screen to be clicked with the mouse | E.g. pause, start buttons etc. Allows for ease of use and saves time especially during break or lunch time where time is limited, as users won’t have to restart the program again |
| Ability to change the amount of rounds in a game or the time limit in the settings | Gives more options to the player on how they would like to optimise their gameplay experience and make it more fun or relaxing for them |
| Start button on starting page brings the player into the character selection page | Makes it easy for the player to instantly start playing and saves students time |
| Energy meter for each character | Allows for a more engaging and fun battle which lasts longer in order to help students to wind down. Must be charged up using a key input during battle so that a certain amount of energy is used for a special attack and an increased amount for ultimate attacks in order to deal more damage to the opponent to defeat them quicker |
| Health bar for each character | Determines which player will win. The player with the least amount of health when the time runs out, or the player who loses all of their health loses. Allows players to see how much health they have remaining so that they can plan out their next moves and keep students minds awake and focussed for lessons |

##### Hardware and software required for the program:

| **Requirement** | **Explanation** |
| --- | --- |
| Standard peripherals: computer or laptop with a keyboard and mouse | The player needs a basic computer to be able to play and navigate the game. A keyboard is needed in order to use the different keys to be inputted to perform different attacks and moves. A mouse or touchpad will be needed in order to click the buttons on screen |
| Windows, Mac or Linux operating system | These are the operating systems that are supported by Python, as the code will be written in Python |
| Python with Tkinter | The program will run on Python using the Tkinter GUI |

##### Success criteria for the program:

| **Criteria** | **How to evidence** |
| --- | --- |
| Main starting page with start button, instructions on how to play, settings | Screenshot of the main starting page that shows where the ‘How to play’ and settings button are |
| Engaging and vibrant design | Screenshots of the windows with bright colours and appealing fonts, characters and world design |
| Visible pause button during battle with pause symbol | Screenshot of the battle screen with a pause button visible and not drowned out by the background |
| Settings menu on a separate page | Screenshot of the settings menu that is separate from the starting page, showing the ability to change the amount of rounds and the time limit in a game |
| Instructions on a separate page, shows which keys are used for player 1 and player 2 | Screenshot of the instructions of how to play separate from the starting page, detailing the movement (jump, left, right, guard), normal attack, special attack and ultimate attack buttons for both player one and player two |
| Character selection on a separate page | Screenshot of the character selection with the option to play against a computer or against another player in character selection and a button to press to change the second player (opponent) into player two, where the default is against a computer |

##### 

##### Limitations:

Potential limitations of the game may include that the game can only be run on a computer or laptop with Python installed. This limits the amount of people who can play the game, especially students who might want to play the game at home but do not have a PC with Python. Another limitation is that the game will only be available in English. Though the game will mostly be playable even without knowledge of English, some players may find the health and energy bars confusing if they have no prior knowledge of similar fighting games with similar bars.

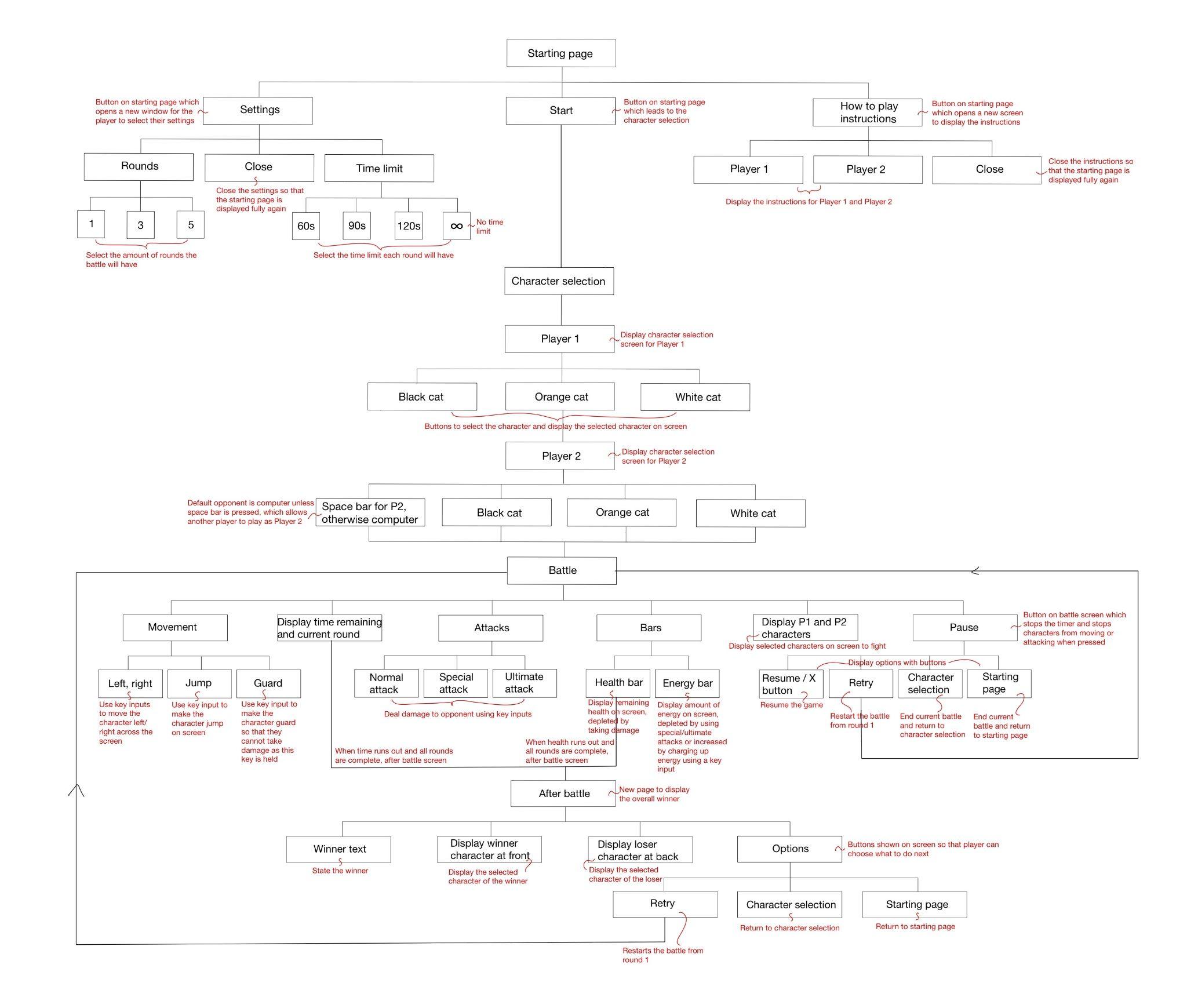
##### 

### Design of the solution

#### Decompose the solution

Using a top down design diagram, I have decomposed the solution into small sub-programs to be coded in the game, shown below, and have used this to detail the class diagrams to code the solution using object-oriented programming, using flowcharts to demonstrate how the algorithm will work.

##### Top down design diagram



##### Battle

|  | **Orange** | **Description** | **Explanation** |
| --- | --- | --- | --- |
| **Attributes** | - image | Stores the image of the character | The selected character image is displayed on the battle screen |
| - x | Stores the x location of the character | To be used when making the character move and jump |
| - y | Stores the y location of the character | To be used when making the character move and jump |
| + damage1 | Stores the damage of the character | The correct damage amount depletes the opponent’s health when an attack is performed |
| + guarding1 | Checks if the character is guarding | The character cannot take damage when guarding |
| **Methods** | + move\_left() | Moves the character image left by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + move\_right() | Moves the character image right by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + getLeft() | Returns the coordinates of the left side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getTop() | Returns the coordinates of the top side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getRight() | Returns the coordinates of the right side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getBottom() | Returns the coordinates of the bottom side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + jump() | Makes the character image jump by 50 pixels | Makes it easier for players to evade attacks or get closer to their opponent to attack |
| + guard() | Makes the character guard so they cannot take damage | Creates a more interesting battle experience as players can strategise and time when they should guard or attack |
| + unguard() | Makes the character unguard so they can take damage | The character is able to attack their opponent again |
| + calcEnergy() | Increases the character’s energy by 10 if their energy does not equal 500, otherwise their energy remains at 500 | Charges up the character’s energy so that they can perform special or ultimate attacks against their opponent to inflict more damage |
| + normalAtk() | Stores 20 in the attribute damage1 and inflicts this amount of damage onto the opponent | Lower damage attack that does not require energy so that players can still attack even if they don’t have any energy remaining |
| + specialAtk() | If the character’s energy is greater than or equal to 50, consumes 50 energy and stores 70 in the attribute damage1 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + ultAtk() | If the character’s energy is greater than or equal to 100, consumes 100 energy and stores 135 in the attribute damage1 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + takeDamage() | If the character’s health is greater than the damage inflicted upon them, their health is decreased by the amount of damage inflicted upon them, otherwise their health will equal 0 | Makes the character take damage that is inflicted upon them by the opponent and ensures the character is defeated when their health drops to 0 |

|  | **Orange2** | **Description** | **Explanation** |
| --- | --- | --- | --- |
| **Attributes** | - image | Stores the image of the character | The selected character image is displayed on the battle screen |
| - x | Stores the x location of the character | To be used when making the character move and jump |
| - y | Stores the y location of the character | To be used when making the character move and jump |
| + damage2 | Stores the damage of the character | The correct damage amount depletes the opponent’s health when an attack is performed |
| + guarding2 | Checks if the character is guarding | The character cannot take damage when guarding |
| **Methods** | + move\_left() | Moves the character image left by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + move\_right() | Moves the character image right by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + getLeft() | Returns the coordinates of the left side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getTop() | Returns the coordinates of the top side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getRight() | Returns the coordinates of the right side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getBottom() | Returns the coordinates of the bottom side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + jump() | Makes the character image jump by 50 pixels | Makes it easier for players to evade attacks or get closer to their opponent to attack |
| + guard() | Makes the character guard so they cannot take damage | Creates a more interesting battle experience as players can strategise and time when they should guard or attack |
| + unguard() | Makes the character unguard so they can take damage | The character is able to attack their opponent again |
| + calcEnergy() | Increases the character’s energy by 10 if their energy does not equal 500, otherwise their energy remains at 500 | Charges up the character’s energy so that they can perform special or ultimate attacks against their opponent to inflict more damage |
| + normalAtk() | Stores 20 in the attribute damage2 and inflicts this amount of damage onto the opponent | Lower damage attack that does not require energy so that players can still attack even if they don’t have any energy remaining |
| + specialAtk() | If the character’s energy is greater than or equal to 50, consumes 50 energy and stores 70 in the attribute damage2 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + ultAtk() | If the character’s energy is greater than or equal to 100, consumes 100 energy and stores 135 in the attribute damage2 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + takeDamage() | If the character’s health is greater than the damage inflicted upon them, their health is decreased by the amount of damage inflicted upon them, otherwise their health will equal 0 | Makes the character take damage that is inflicted upon them by the opponent and ensures the character is defeated when their health drops to 0 |

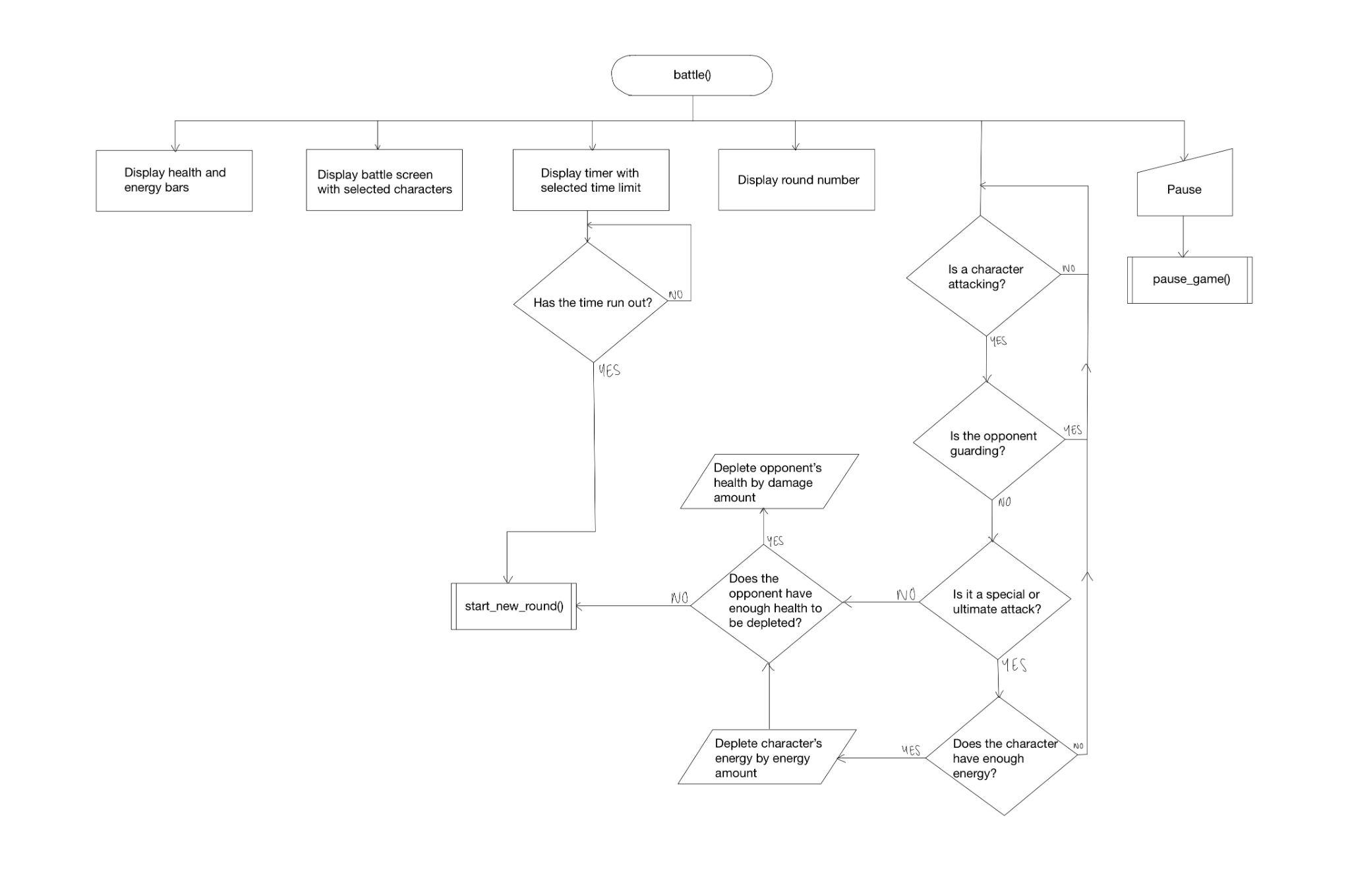
|  | **White** | **Description** | **Explanation** |
| --- | --- | --- | --- |
| **Attributes** | - image | Stores the image of the character | The selected character image is displayed on the battle screen |
| - x | Stores the x location of the character | To be used when making the character move and jump |
| - y | Stores the y location of the character | To be used when making the character move and jump |
| - damage1 | Stores the damage of the character | The correct damage amount depletes the opponent’s health when an attack is performed |
| + guarding1 | Checks if the character is guarding | The character cannot take damage when guarding |
| **Methods** | + move\_left() | Moves the character image left by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + move\_right() | Moves the character image right by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + getLeft() | Returns the coordinates of the left side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getTop() | Returns the coordinates of the top side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getRight() | Returns the coordinates of the right side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getBottom() | Returns the coordinates of the bottom side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + jump() | Makes the character image jump by 50 pixels | Makes it easier for players to evade attacks or get closer to their opponent to attack |
| + guard() | Makes the character guard so they cannot take damage | Creates a more interesting battle experience as players can strategise and time when they should guard or attack |
| + unguard() | Makes the character unguard so they can take damage | The character is able to attack their opponent again |
| + calcEnergy() | Increases the character’s energy by 10 if their energy does not equal 500, otherwise their energy remains at 500 | Charges up the character’s energy so that they can perform special or ultimate attacks against their opponent to inflict more damage |
| + normalAtk() | Stores 35 in the attribute damage1 and inflicts this amount of damage onto the opponent | Lower damage attack that does not require energy so that players can still attack even if they don’t have any energy remaining |
| + specialAtk() | If the character’s energy is greater than or equal to 50, consumes 50 energy and stores 60 in the attribute damage1 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + ultAtk() | If the character’s energy is greater than or equal to 100, consumes 100 energy and stores 130 in the attribute damage1 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + takeDamage() | If the character’s health is greater than the damage inflicted upon them, their health is decreased by the amount of damage inflicted upon them, otherwise their health will equal 0 | Makes the character take damage that is inflicted upon them by the opponent and ensures the character is defeated when their health drops to 0 |

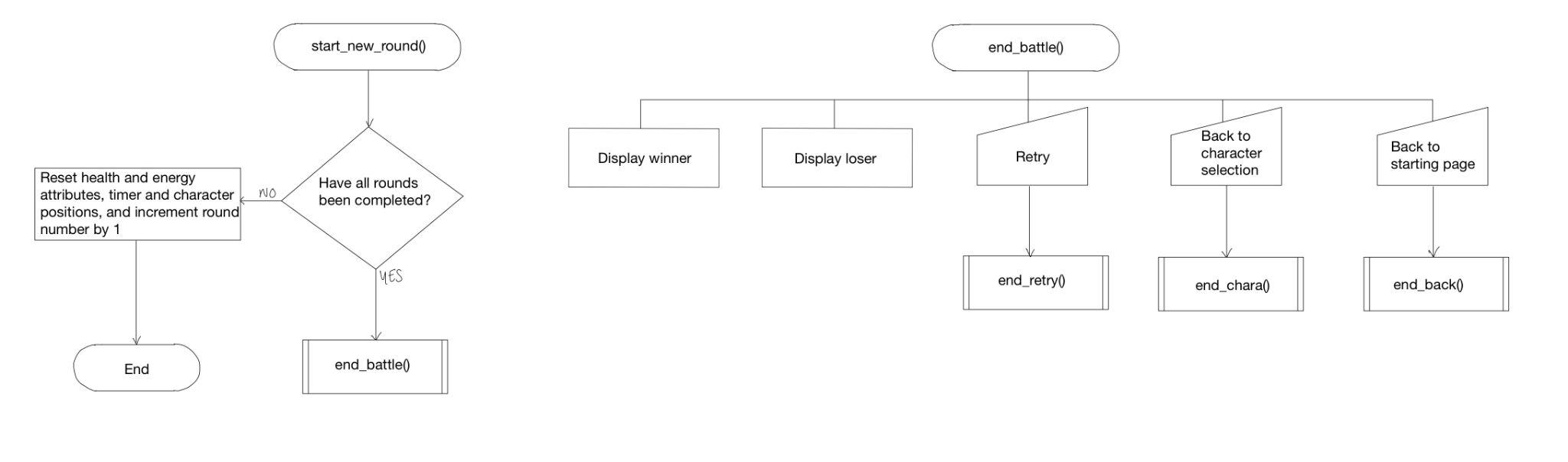
|  | **White2** | **Description** | **Explanation** |
| --- | --- | --- | --- |
| **Attributes** | - image | Stores the image of the character | The selected character image is displayed on the battle screen |
| - x | Stores the x location of the character | To be used when making the character move and jump |
| - y | Stores the y location of the character | To be used when making the character move and jump |
| + damage2 | Stores the damage of the character | The correct damage amount depletes the opponent’s health when an attack is performed |
| + guarding2 | Checks if the character is guarding | The character cannot take damage when guarding |
| **Methods** | + move\_left() | Moves the character image left by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + move\_right() | Moves the character image right by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + getLeft() | Returns the coordinates of the left side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getTop() | Returns the coordinates of the top side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getRight() | Returns the coordinates of the right side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getBottom() | Returns the coordinates of the bottom side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + jump() | Makes the character image jump by 50 pixels | Makes it easier for players to evade attacks or get closer to their opponent to attack |
| + guard() | Makes the character guard so they cannot take damage | Creates a more interesting battle experience as players can strategise and time when they should guard or attack |
| + unguard() | Makes the character unguard so they can take damage | The character is able to attack their opponent again |
| + calcEnergy() | Increases the character’s energy by 10 if their energy does not equal 500, otherwise their energy remains at 500 | Charges up the character’s energy so that they can perform special or ultimate attacks against their opponent to inflict more damage |
| + normalAtk() | Stores 35 in the attribute damage2 and inflicts this amount of damage onto the opponent | Lower damage attack that does not require energy so that players can still attack even if they don’t have any energy remaining |
| + specialAtk() | If the character’s energy is greater than or equal to 50, consumes 50 energy and stores 60 in the attribute damage2 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + ultAtk() | If the character’s energy is greater than or equal to 100, consumes 100 energy and stores 130 in the attribute damage2 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + takeDamage() | If the character’s health is greater than the damage inflicted upon them, their health is decreased by the amount of damage inflicted upon them, otherwise their health will equal 0 | Makes the character take damage that is inflicted upon them by the opponent and ensures the character is defeated when their health drops to 0 |

|  | **Black** | **Description** | **Explanation** |
| --- | --- | --- | --- |
| **Attributes** | - image | Stores the image of the character | The selected character image is displayed on the battle screen |
| - x | Stores the x location of the character | To be used when making the character move and jump |
| - y | Stores the y location of the character | To be used when making the character move and jump |
| + damage1 | Stores the damage of the character | The correct damage amount depletes the opponent’s health when an attack is performed |
| + guarding1 | Checks if the character is guarding | The character cannot take damage when guarding |
| **Methods** | + move\_left() | Moves the character image left by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + move\_right() | Moves the character image right by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + getLeft() | Returns the coordinates of the left side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getTop() | Returns the coordinates of the top side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getRight() | Returns the coordinates of the right side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getBottom() | Returns the coordinates of the bottom side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + jump() | Makes the character image jump by 50 pixels | Makes it easier for players to evade attacks or get closer to their opponent to attack |
| + guard() | Makes the character guard so they cannot take damage | Creates a more interesting battle experience as players can strategise and time when they should guard or attack |
| + unguard() | Makes the character unguard so they can take damage | The character is able to attack their opponent again |
| + calcEnergy() | Increases the character’s energy by 10 if their energy does not equal 500, otherwise their energy remains at 500 | Charges up the character’s energy so that they can perform special or ultimate attacks against their opponent to inflict more damage |
| + normalAtk() | Stores 25 in the attribute damage1 and inflicts this amount of damage onto the opponent | Lower damage attack that does not require energy so that players can still attack even if they don’t have any energy remaining |
| + specialAtk() | If the character’s energy is greater than or equal to 50, consumes 50 energy and stores 50 in the attribute damage1 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + ultAtk() | If the character’s energy is greater than or equal to 100, consumes 100 energy and stores 150 in the attribute damage1 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + takeDamage() | If the character’s health is greater than the damage inflicted upon them, their health is decreased by the amount of damage inflicted upon them, otherwise their health will equal 0 | Makes the character take damage that is inflicted upon them by the opponent and ensures the character is defeated when their health drops to 0 |

|  | **Black2** | **Description** | **Explanation** |
| --- | --- | --- | --- |
| **Attributes** | - image | Stores the image of the character | The selected character image is displayed on the battle screen |
| - x | Stores the x location of the character | To be used when making the character move and jump |
| - y | Stores the y location of the character | To be used when making the character move and jump |
| + damage2 | Stores the damage of the character | The correct damage amount depletes the opponent’s health when an attack is performed |
| + guarding2 | Checks if the character is guarding | The character cannot take damage when guarding |
| **Methods** | + move\_left() | Moves the character image left by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + move\_right() | Moves the character image right by 10 pixels | Allows for quick movement across the screen in order to attack or evade attacks |
| + getLeft() | Returns the coordinates of the left side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getTop() | Returns the coordinates of the top side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getRight() | Returns the coordinates of the right side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + getBottom() | Returns the coordinates of the bottom side of the character image | To be used when calculating the distance between the two players so that the opponent only takes damage if the attacker is close enough to them |
| + jump() | Makes the character image jump by 50 pixels | Makes it easier for players to evade attacks or get closer to their opponent to attack |
| + guard() | Makes the character guard so they cannot take damage | Creates a more interesting battle experience as players can strategise and time when they should guard or attack |
| + unguard() | Makes the character unguard so they can take damage | The character is able to attack their opponent again |
| + calcEnergy() | Increases the character’s energy by 10 if their energy does not equal 500, otherwise their energy remains at 500 | Charges up the character’s energy so that they can perform special or ultimate attacks against their opponent to inflict more damage |
| + normalAtk() | Stores 25 in the attribute damage2 and inflicts this amount of damage onto the opponent | Lower damage attack that does not require energy so that players can still attack even if they don’t have any energy remaining |
| + specialAtk() | If the character’s energy is greater than or equal to 50, consumes 50 energy and stores 50 in the attribute damage2 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + ultAtk() | If the character’s energy is greater than or equal to 100, consumes 100 energy and stores 150 in the attribute damage2 and inflicts this amount of damage onto the opponent, otherwise returns false if they do not have enough energy | Inflicts a greater amount of damage upon the opponent to defeat them quicker |
| + takeDamage() | If the character’s health is greater than the damage inflicted upon them, their health is decreased by the amount of damage inflicted upon them, otherwise their health will equal 0 | Makes the character take damage that is inflicted upon them by the opponent and ensures the character is defeated when their health drops to 0 |

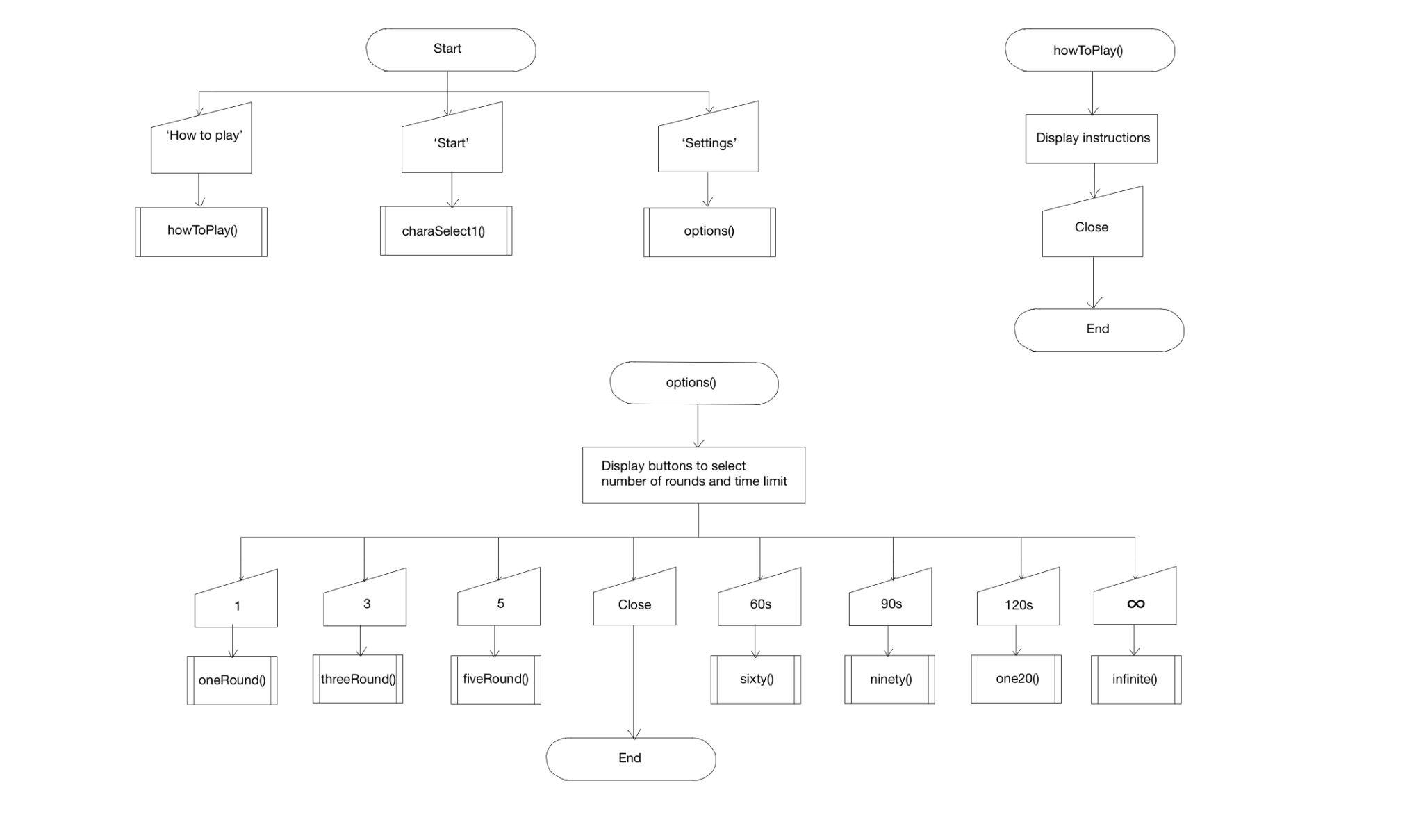
|  | **Game** | **Description** | **Explanation** |
| --- | --- | --- | --- |
| **Attributes** | - master | Stores the game screen | The game will be displayed on this screen |
| - bg | Stores the background image of the starting page | This image will be displayed on the starting page when the program is run |
| + timeLimit | Stores the default time limit (90) | Sets the default time limit as 90 in case the player does not choose a time limit |
| + rounds | Stores the default amount of rounds (3) | Sets the default number of rounds to 3 in case the player does not choose an amount of rounds |
| + selection1 | Stores the default character selection to be validated (0) | Ensures the player does not start the game without selecting a character |
| + selection2 | Stores the default character selection to be validated (0) | Ensures the player does not start the game without selecting a character |
| + winners | Stores the winner of each round in the list | To calculate the overall winner after all rounds have been completed |
| + actualWinner | Stores the actual winner after all rounds are completed | Counts how many times each player has won by how many times they have been stored in the list in order to calculate the overall winner |
| - settings | Stores the settings button | Can be clicked on the starting screen to display a window where the player can choose the time limit they would like each round to be and the amount of rounds they would like to play to allow more choice of how players would like to play and relax |
| - instructions | Stores the instructions button | Can be clicked on the starting screen to display a window that shows each key input needed to play the game and what they do so that players know how to play |
| - start | Stores the start button | Clicked on the starting screen to immediately take the player to the character selection for easy and quick use |
| **Methods** | + howToPlay() | Creates a new window which displays the instructions for each player on how to play | Displays a window that shows each key input needed to play the game and what they do so that players know how to play |
| + options() | Creates a new window which has different buttons for the amount rounds and the amount of time each round will be | Displays a window where the player can choose the time limit they would like each round to be and the amount of rounds they would like to play to allow more choice of how players would like to play and relax |
| + oneRound() | Stores 1 in the attribute rounds | Only 1 round is played when selected by the player |
| + threeRound() | Stores 3 in the attribute rounds | 3 rounds are played when selected by the player |
| + fiveRound() | Stores 5 in the attribute rounds | 5 rounds are played when selected by the player |
| + sixty() | Stores 60 in the attribute timeLimit | Each round is 60 seconds when selected by the player |
| + ninety() | Stores 90 in the attribute timeLimit | Each round is 90 seconds when selected by the player |
| + one20() | Stores 120 in the attribute timeLimit | Each round is 120 seconds when selected by the player |
| + infinite() | Stores ‘infinite’ in the attribute timeLimit | Each round has no time limit when selected by the player |
| + charaSelect1() | Displays three buttons with the three different characters, where the user selects a button in order to choose a character for the first player | Allows the player to select a character to allow for more choice and so that players do not get mixed up with their characters on the battle screen |
| + charaSelect2() | Displays three buttons with the three different characters, where the user selects a button in order to choose a character for the second player | Allows the player to select a character to allow for more choice and so that players do not get mixed up with their characters on the battle screen |
| + oChar1() | Stores 1 in the attribute selection1 when the user presses the Orange Cat button, which then displays the character details to the user | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |
| + oChar2() | Stores 1 in the attribute selection2 when the user presses the Orange Cat button, which then displays the character details to the user | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |
| + wChar1() | Stores 2 in the attribute selection1 when the user presses the White Cat button, which then displays the character details to the user | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |
| + wChar2() | Stores 2 in the attribute selection2 when the user presses the White Cat button, which then displays the character details to the user | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |
| + bChar1() | Stores 3 in the attribute selection1 when the user presses the Black Cat button, which then displays the character details to the user | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |
| + bChar2() | Stores 3 in the attribute selection2 when the user presses the Black Cat button, which then displays the character details to the user | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |
| + battle() | Starts the battle | Displays the battle screen with the correct characters, time limit and round number on screen |
| + start\_timer() | Starts the timer | Counts down to 0 so that if a character isn’t defeated, the next round is started (or the battle is ended if all rounds are complete) |
| + update\_timer() | Updates the timer on screen | Counts down to 0 so that if a character isn’t defeated, the next round is started (or the battle is ended if all rounds are complete) |
| + pause\_game() | Pauses the game | Stops the characters from moving or attacking and pauses the timer, and displays a frame on screen with options the player may like to take |
| + resume\_game() | Resumes the game | Allows the characters to move and attack again, and resumes the timer |
| + show\_pause\_frame() | Shows the pause frame when the pause button is clicked | Displays the options the player may like to take |
| + instructions() | Displays the instructions again when a button is clicked in the pause frame | Reminds the player how to play if they forget or did not check on the starting screen |
| + hide\_pause\_frame() | Hides the pause frame | Removes the pause frame off screen when the game is resumed as it is no longer needed |
| + pause\_timer() | Pauses the timer when the game is paused | Ensures the time does not run out when the player is not playing |
| + resume\_timer() | Resumes the timer when the game is unpaused | Enures the battle continues as normal once the game is unpaused |
| + pause\_characters() | Stops the characters from moving or attacking when the game is paused | Ensures the characters cannot attack their opponent when a player isn’t playing or looking at the instructions for example |
| + resume\_characters() | Allows the characters to move and attack again when the game is unpaused | Enures the battle continues as normal once the game is unpaused |
| + retry() | Restarts the battle again if ‘Retry’ is selected on the pause frame | Players can start again if needed |
| + start\_new\_round() | Starts a new round after the end of one, or otherwise leads to the after battle screen if all rounds are complete | Ensures the selected amount of rounds is played |
| + end\_battle() | Displays the after battle screen with the winners, as well as the options to retry, go back to character selection or go back to the starting page | Makes sure the players know who has won and gives them options to choose from of what they would like to do next |
| + end\_retry() | Restarts the battle again if ‘Retry’ is selected on the after battle screen | Players can play again with the current selected characters and settings if the would like |
| + end\_chara() | Goes back to the character selection if ‘Character selection’ is selected in the after battle screen | Players can choose different characters to play again |
| + end\_back() | Goes back to the starting page if ‘Starting page’ is selected in the after battle screen | Players can change their settings on the starting screen to have a different battle experience |
| + battle\_chara() | Goes back to the character selection if ‘Character selection’ is selected in the pause frame | Players can choose different characters if they would like to switch their characters |
| + battle\_back() | Goes back to the starting page if ‘Starting page’ is selected in the pause frame | Players can change their settings on the starting screen if they would like to change them |





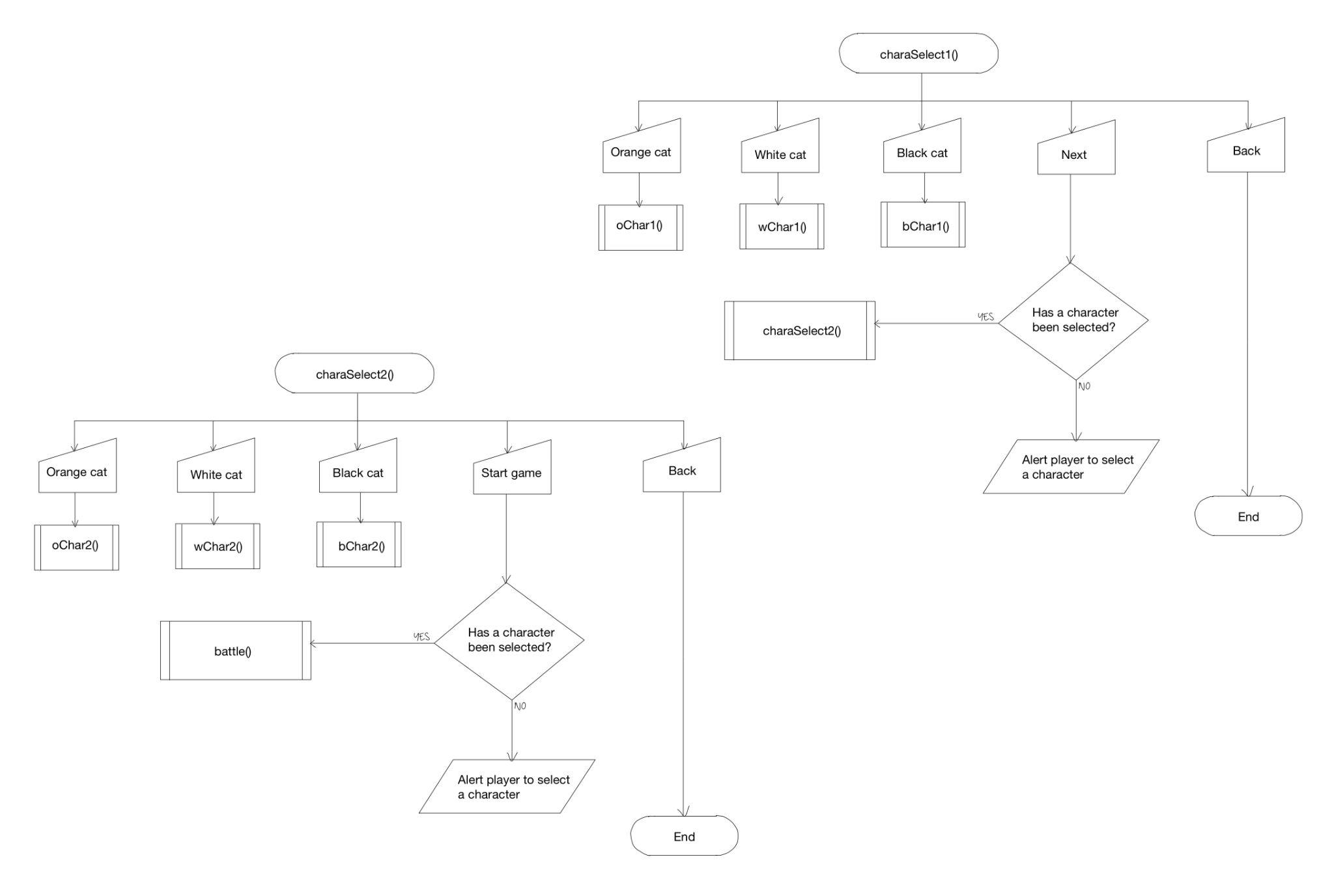
##### Starting page

| **Attribute name** | **Type** | **Size (range of possible values)** | **Description** | **Explanation** |
| --- | --- | --- | --- | --- |
| + timeLimit | Integer | 60, 90, 120 | Stores the value of the time selected in the settings (this determines the length of time each round will be) | Ensures the player’s choice of time limit is stored and displayed on the battle screen |
| + rounds | Integer | 1, 3, 5 | Stores the value of the rounds selected in the settings (this determines the amounts of rounds the battle will have) | Ensures the player’s choice of the amount of rounds they would like to play is stored so that that amount of rounds is played |



##### Character Selection

| **Attribute name** | **Type** | **Size (range of possible values)** | **Description** | **Explanation** |
| --- | --- | --- | --- | --- |
| + selection1 | Integer | 0,1,2,3 | Stores the value of the character selection for Player 1 in the character selection (selection1=1 for Orange Cat, selection1=2 for White Cat, selection1=3 for Black Cat) | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |
| + selection2 | Integer | 0,1,2,3 | Stores the value of the character selection for Player 2 in the character selection (selection2=1 for Orange Cat, selection2=2 for White Cat, selection2=3 for Black Cat) | Ensures the player’s character selection is stored so that the correct character is displayed on the battle screen |



#### Describe the solution

##### Design objectives

* + The design objectives are similar to the requirements specified previously. The game will start with a vibrant and enticing starting page as preferred by potential users, with three different buttons: a settings button, which opens a new window with buttons to change the amount of rounds in a battle and buttons to change how long the rounds will be; a ‘How to Play’ button, which opens a new window and displays in a simple format the key inputs needed for both Player 1 and Player 2 so that players know how to play; and a start button, which leads to the character selection for Player 1 for quick and easy use.
  + The character selection will have a selection of three different characters, displaying their different attacks and their damage amounts in a box so users have more choice in what character they would like to play. This would then lead to the character selection for Player 2, displayed similarly, and then this would lead into the battle.
  + The battle will display the characters selected by the players, as well as their health and energy bars at the top of the screen and the timer and the rounds remaining between each players’ bars so that players are aware of how much time, health and energy they have remaining and can strategise their next moves from there. When the time runs out or a player is defeated, it will lead to the after battle page, where the players will have three different options: retry, return to character selection and return to starting page, which gives them the choice of whether they would like to play again with the same characters or settings or change them.

##### Aesthetic considerations

* + The screen size will be moderately sized in order to fit most screens (863x600 pixels)
  + There will be three different characters: an orange cat, a black cat and a white cat to give players a choice of what character they would like to play as well as make sure characters don’t get mixed up on the battle screen
  + Each page within the game will follow a similar colour scheme of blue, orange, pink, purple and green for a vibrant design that was preferred by potential users
  + In battle, Player 1 will be facing right and Player 2 will be facing left so that players can easily tell who is who
  + In battle, the characters will be approximately 100x120 pixels to make sure they are not too big or too small on the screen size
  + When moving, the characters will move 10 pixels at a time so that characters can move quickly across the screen to attack or evade attacks

##### Input considerations

* + The player will be able to choose how many rounds they would like to play to give them more choice to play in a way that will help them to destress
  + The player will be able to choose how much time they would like the round(s) to be to give them more choice to play in a way that will help them to destress
  + The players will be able to choose which character they would like to play as to give players a choice of what character they would like to play as well as make sure characters don’t get mixed up on the battle screen
  + The players will be able to control the movement of the characters left, right and up using the arrow keys (Player 2) and WASD keys (Player 1) as these are keys that are commonly used to move characters
  + The players will be able to use key inputs to perform three different attacks on their opponents: normal attack, special attack and ultimate attack, as well as to recharge their energy and have the ability to guard to give a more enticing battle experience that lasts longer
  + The player will be able to view a ‘How to Play’ screen that opens in a new window by clicking a button that is on the initial starting screen so that players can easily access the instructions
  + The player will be able to view and change the settings that opens in a new window by clicking a button that is on the initial starting screen so that players can easily change their settings before starting the battle
  + The player will be able to go back to the initial starting screen from the character selection screen and will be able to go back to initial starting screen, character selection screen or battle after a battle has ended so that players can change their characters or settings if they would like to
  + The players will be able to pause the battle, and through this, will be able to retry, go back to the character selection or go back to the starting page, which gives players the choice of what they would like to do rather than just having a resume button

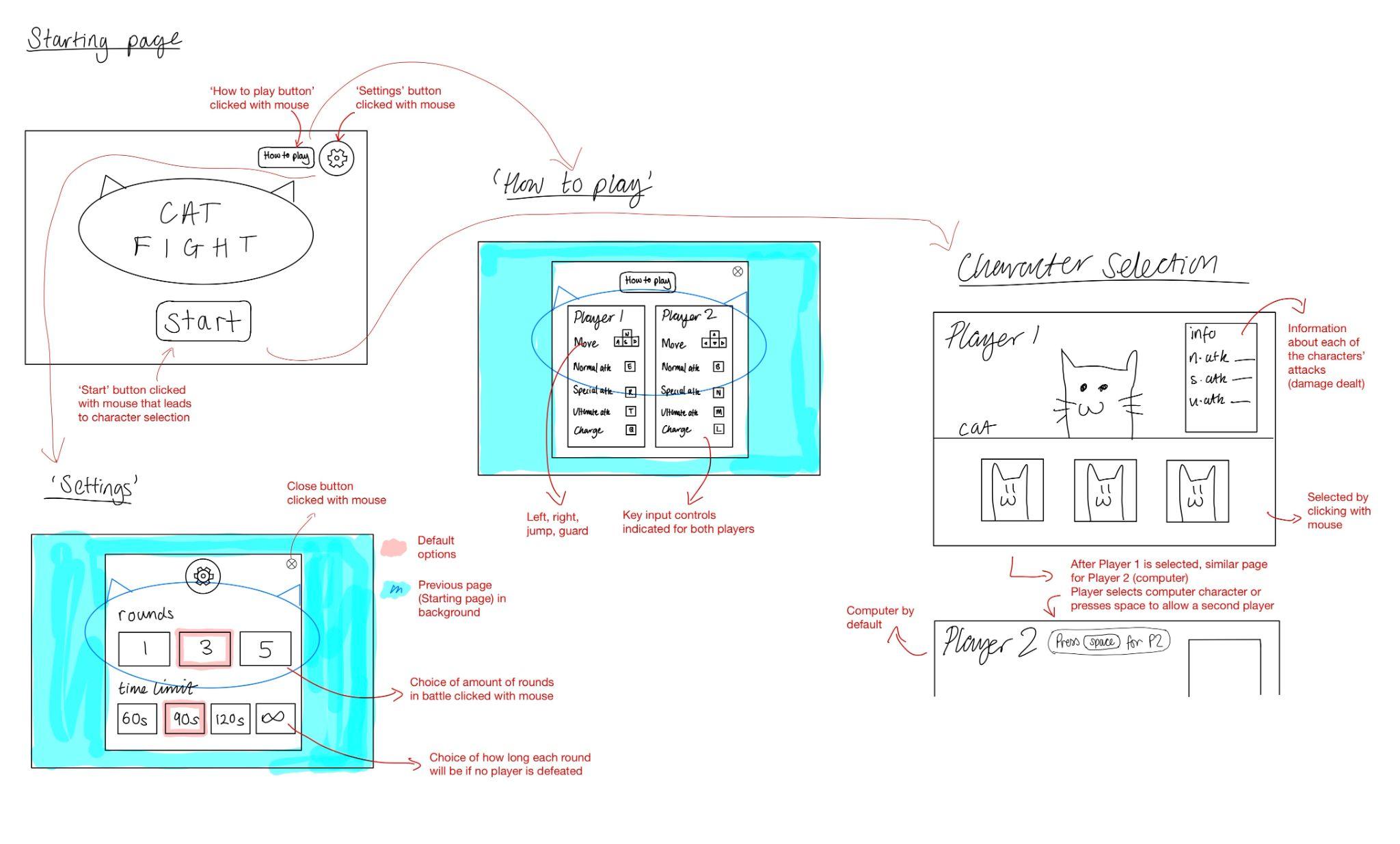
##### Processing considerations

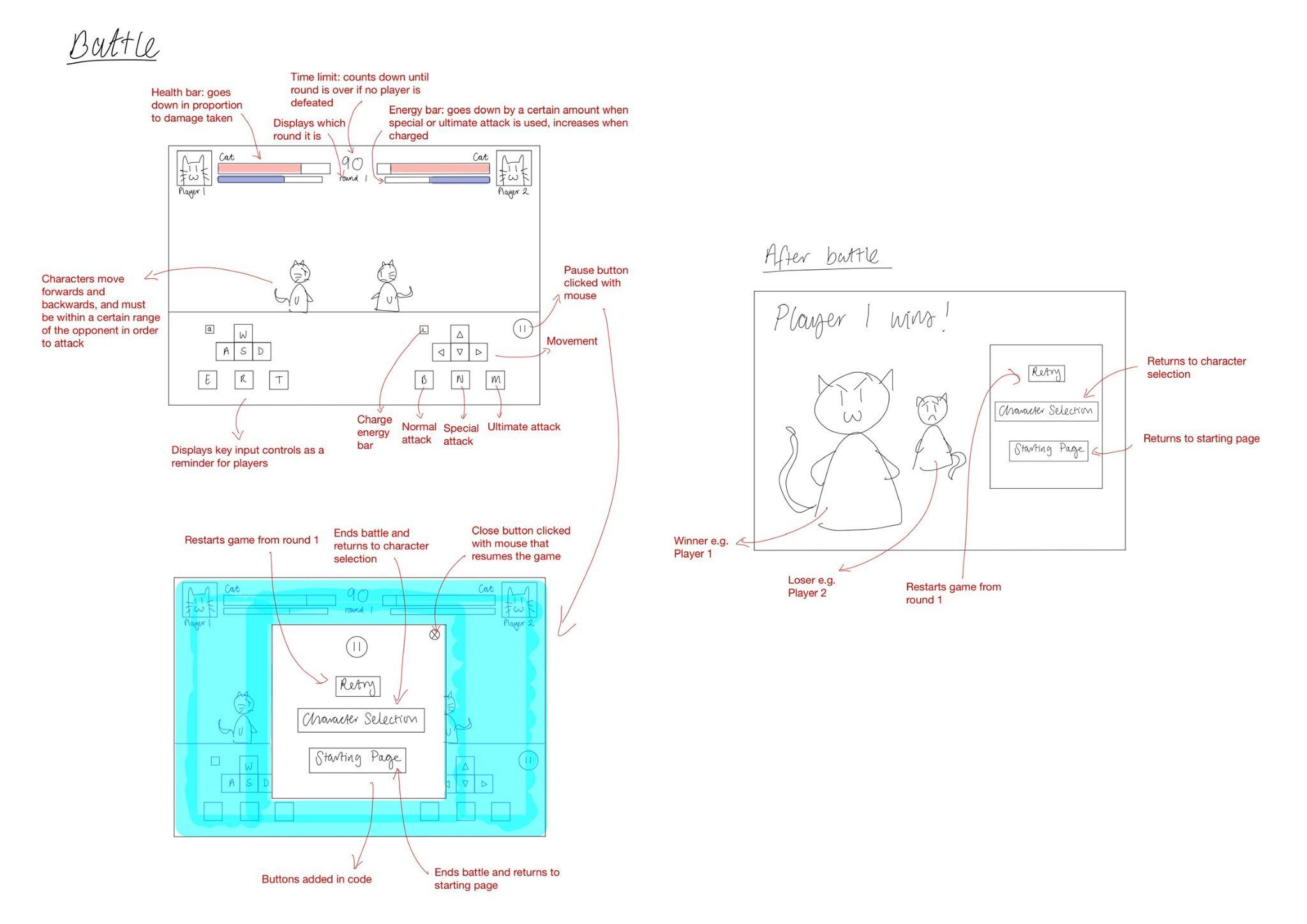
* + When a character is attacked, their health will go down by the amount of damage dealt by the attack
  + When a character uses a special attack or an ultimate attack, their energy will go down by a certain amount
  + When a player uses a key input to recharge their energy, their energy will go up by 10
  + The characters must not be able to go off-screen to ensure that the battle only happens on-screen
  + The characters should only be able to attack within a certain distance of their opponent, which ensures that players cannot continuously keep attacking each other
  + The battle will end when a character is defeated (their health is equal to 0), when time runs out, or when the players manually end it through the pause button options
  + The winner of each round must be recorded in order to declare a final winner by the end of all of the rounds

##### Output considerations

* + The current round will be displayed
  + The time remaining will be displayed
  + The amount of health each player has remaining will be displayed
  + The amount of energy each player has will be displayed
  + Which characters are playing that have been chosen by the players will be displayed
  + Instructions on how to play the game will be displayed during battle and also through a button on the starting screen

##### Interface design







#### Describe the approach to testing

I have created a table below which lays out which parts of my code are to be tested throughout the development process.

| **Function being tested** | **Method of input** | **Input** | **Expected result** |
| --- | --- | --- | --- |
| Character selection | Button clicked on the character selection screen | Orange Cat button clicked (selection(1 or 2) = 1) | The orange cat character will appear on the battle screen to be played by the user |
| White Cat button clicked (selection = 2) | The white cat character will appear on the battle screen to be played by the user |
| Black Cat button clicked (selection = 3) | The black cat character will appear on the battle screen to be played by the user |
| No button clicked | The user will prompted with an alert message to select a character before moving onto the next screen |
| Character movement | Pressing a button on the keyboard | A | Player 1 will move left |
| D | Player 1 will move right |
| W | Player 1 will jump |
| S | Player 1 will guard (They will not be able to take damage) |
| Left | Player 2 will move left |
| Right | Player 2 will move right |
| Up | Player 2 will jump |
| Down | Player 2 will guard (They will not be able to take damage) |
| Collision with borders | When in battle, head into a border of the game screen using the directional buttons on the keyboard | D (Player 1)/Right (Player 2) to direct the characters to the right border of the game screen | The characters will not be able to move past the borders of the game screen and will stop moving at the border. |
| A (Player 1)/Left (Player 2) to direct the characters to the left border of the game screen |
| Collision with opponent | When in battle, head into the direction of the opposing player using the directional buttons on the keyboard | Directional buttons on the keyboard used to direct the characters to their opponents | The characters will not be able to move past each other and will stop moving when the border of one character touches the border of the other |
| Character using normal attack | Pressing a button on the keyboard | E (Player 1)/ B (Player 2) | Damage will be dealt to the opponent by the normal attack damage amount of the player and the opponent’s health will decrease by this much |
| Character using special attack | Pressing a button on the keyboard | R (Player 1)/ N (Player 2) | The player’s energy will decrease by 50 and damage will be dealt to the opponent by the special attack damage amount of the player, decreasing the opponent’s health by this much |
| Character using ultimate attack | Pressing a button on the keyboard | T (Player 1)/ M (Player 2) | The player’s energy will decrease by 100 and damage will be dealt to the opponent by the special attack damage amount of the player, decreasing the opponent’s health by this much |
| Character recharging energy | Pressing a button on the keyboard | Q (Player 1)/ L (Player 2) | The player’s energy will increase by 10 if the energy bar is not full, otherwise their energy will remain at 500 |
| Displaying instructions | Clicking the ‘How to play’ button on the starting screen | N/A | Displays a new window with the instructions on the key inputs for both Player 1 and Player 2 |
| Displaying settings | Clicking the ‘Settings’ button on the starting screen | N/A | Displays a new window with the options for the amount of rounds and the time limit in battle |
| Selecting amount of rounds | Button clicked in settings | ‘1’ button clicked (rounds = 1) | One round will be played during battle |
| ‘3’ button clicked (rounds = 3) | Three rounds will be played during battle, and the player who wins the most rounds will win |
| ‘5’ button clicked (rounds = 5) | Five rounds will be played during battle, and the player who wins the most rounds will win |
| No button clicked | The rounds will be set to a default of three rounds |
| Selecting the time limit of the rounds | Button clicked in settings | ‘60’ button clicked | Each round will have timer of 60 seconds |
| ‘90’ button clicked | Each round will have timer of 90 seconds |
| ‘120’ button clicked | Each round will have timer of 120 seconds |
| ‘∞’ button clicked | Each round will continue for an infinite amount of time and will only end when a character is defeated |
| No button clicked | The default time limit will be set to 90 seconds |
| Selecting the pause button | Button clicked during battle | N/A | Pauses the game so that characters are unable to move or attack and displays a window with options to retry, return to character selection or return to the starting page, and can cross off the window to resume |
| Selecting the ‘Retry’ button | Button clicked after pausing or after battle | N/A | Restarts the battle from the start with full energy and health bars |
| Selecting the ‘Character Selection’ button | Button clicked after pausing or after battle | N/A | Ends the current battle and returns to the character selection |
| Selecting the ‘Starting Page’ button | Button clicked after pausing or after battle | N/A | Ends the current battle and returns to the starting page |

### 

### Developing the solution

#### Iterative development process

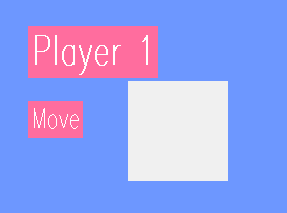
To begin the development process, I used my top down design structure on page 15 to break down this process into different key stages, which I will describe below, highlighting any problems I faced and how they were resolved, as well as indicating which test each part of the development corresponds to in the testing section on page 75.

##### 1) Create the starting page

Firstly, I made a basic starting interface including a settings button with the ability to change the time limit and the amount of rounds in battle, a ‘How to play’ button which displays the key input controls for each player, and a start button which takes the player to character selection.

For the settings button, I created a subroutine named options as shown partially below, which displays a new window with more buttons to change the time limit and the rounds (Test 9). Each button has a command in order to run a different subroutine, for example if the user were to choose a sixty second time limit and press the button ‘60’, the subroutine time60 would run (line 18), which would store the integer 60 in the global variable timeLimit (line 20) (Test 11). I made timeLimit a global variable so that it could be used later in order to implement the player’s choice of the time limit. At first, I implemented the settings through Tkinter’s Toplevel widget, which created a new window, however I realised that the integers in the timeLimit and rounds variables didn’t store correctly, therefore I decided to display the settings in a frame on screen and hid the frame with a close button (line 19) using Tkinter’s pack\_forget method, which was closer to my initial design.

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | def options():  sett = Frame(root, bg='#6D97FF')  sett.pack(fill="both", expand="yes")  top = Label(sett, text = "Settings", bg='#FF6D9E',  fg='white', font=('Modern 30 bold'))  top.pack(pady=20)    timeLimit = Label(sett, text="Time limit",  bg='#FF6D9E',  fg='white', font=('Modern 25 bold'))  timeLimit.pack(pady=120)  time60 = Button(sett, text='60s', fg='white',  bg='#FF6D9E',  bd=1, font=('Modern 15 bold'),  command = sixty, height=2, width=8)  time60.place(x=160,y=440)  close = Button(sett, text='X', fg='white',  bg='#FF746C', bd=1, font=('Modern 20 bold'),  command = sett.pack\_forget, height=1, width=4)  close.place(x=750, y=20)    def sixty():  global timeLimit  timeLimit = 60 |
| --- | --- |

For the ‘How to play’ button, I created a subroutine named howToPlay, which displays a frame with the controls of each player shown (Test 8). Initially, I struggled to make images of the key input controls I wanted to display show up, and what would come up was a blank space in the place of the image as shown below using the code below.

| 01  02  03 | wasd = PhotoImage(file="wasd.png")  wasd\_label = Label(inst, image=wasd, bd=0)  wasd\_label.place(x=200, y=180) |
| --- | --- |

After some research, I realised that my error was due to assigning the image ‘wasd’ to the label ‘wasd\_label’ directly in line 02, however in Tkinter, when using the PhotoImage class to load an image, it needs to be retained by keeping a reference to it, otherwise the image won’t display. Therefore, to correct this error, I added a reference to the PhotoImage object using the line: wasd.image = wasd.

For the start button, I created a subroutine called charaSelect1, which takes the player to a new page, for player 1 to select their character.

##### 2) Create the character selection

Next, I made a character selection for player 1 in the charaSelect1 subroutine by coding three different buttons for each of the characters, each with their own subroutine command in order to display the image of the character on screen, along with their character information such as their attack damage, in order to help the user make a better choice of which character they would like to play (Test 1). For example, when clicked, the button named orange with a picture of Orange Cat displayed would perform subroutine oChar1 as shown below, which would make the global variable selection1 equal to 1 (line 03) and display the image of Orange Cat (line 09) and its details (lines 11-22). I made selection1 a global variable so that the user’s choice of character could be retained through an if statement later in the code.

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22 | def oChar1():  global selection1  selection1 = 1    photo = PhotoImage(file="oCS2.png")  photo.image = photo  photo\_label = Label(start, image=photo,  bg='#FF6D9E',bd=0)  photo\_label.place(x=260, y=30)    details = Label(start, text = "Details", bg='#68ffa9',  fg='#DA6BFF', font=('Modern 20 bold'))  details.place(x=600, y=45)  nAtk = Label(start, text = "Normal Attack 20",  bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))  nAtk.place(x=600, y=110)  sAtk = Label(start, text = "Special Attack 70",  bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))  sAtk.place(x=600, y=180)  uAtk = Label(start, text = "Ultimate Attack 135",  bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))  uAtk.place(x=600, y=250) |
| --- | --- |

A ‘Next’ button then leads to the character selection for player 2 in the subroutine charaSelect2, which works the same. The ‘Start game’ button then leads the players into battle through the battle subroutine with the command battle (line 08). The ‘Back’ button takes the player back to the previous screen using Tkinter’s destroy method (line 08).

| 01  02  03  04  05  06  07  08  09 | back = Button(player2, text='Back', fg='#DA6BFF',  bg='#68ffa9', bd=1, font=('Modern 17 bold'),  command = player2.destroy, height=1, width=8)  back.place(x=40,y=160)  toGame = Button(player2, text='Start game', fg='#DA6BFF',  bg='#68ffa9', bd=1, font=('Modern 17 bold'),  command = battle, height=2, width=12)  toGame.place(x=40,y=230) |
| --- | --- |

Similarly to before, I struggled to display the characters’ pictures on the buttons correctly, and they would show up as a blank button. This was similarly rectified by adding a reference to the PhotoImage object.

##### 3) Write the battle code

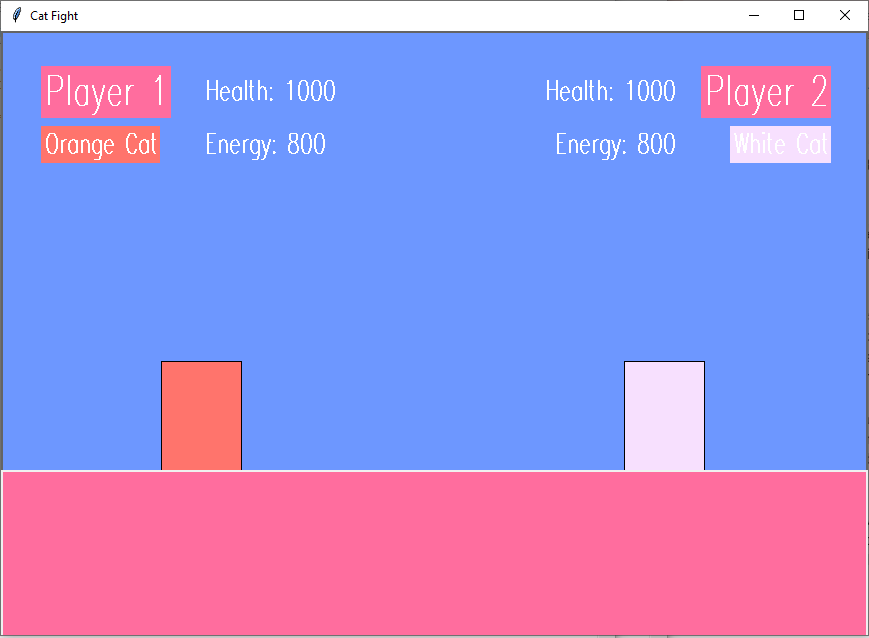
As I had initially written the code for the starting page and character selection using a procedural programming approach, before I started writing the battle code, I decided to use an object-oriented programming approach and therefore coded the battle separately, and decided to add in the rest of the code into the main Game class later to save time and scrolling through code.

For the battle, I had to ensure that only the characters selected by the players would show up on the battle screen. To do this, I made the selection1 attribute for player 1 and the selection2 attribute for player 2, in their respective character selections, public attributes, so that I could use an if statement in the battle method in order to call the correct character object, as shown partially below (line 01).

I made a class for each character as well as a duplicate of them for player 1 and player 2. For example, when selection1 is set to 1, an instance of the Orange class would be instantiated (line 06). When selection2 is set to 1, an instance of the Orange2 class would be instantiated (line 29). I initially only had a single Orange, White and Black class, but then realised this caused problems when it came to binding different keys for both players, as well as when both players selected the same character, therefore I made a duplicate that uses the other player’s keys (lines 11-14 and lines 26-29).

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | if selection1 == 1:  pOrange = Label(play, text = "Orange Cat",  bg='#FF746C', fg='white', font=('Modern 20 bold'))  pOrange.place(x=40, y=95)  self.items = {}  self.orange = Orange(self.canvas, 200, 390)  self.items[self.orange.item] = self.orange  self.hud = None  self.canvas.focus\_set()  self.canvas.bind('<a>',  lambda \_: self.orange.move(-10))  self.canvas.bind('<d>',  lambda \_: self.orange.move(10))  if selection2 == 1:  pOrange = Label(play, text = "Orange Cat",  bg='#FF746C', fg='white', font=('Modern 20 bold'))  pOrange.place(x=712, y=95)  self.items = {}  self.orange2 = Orange2(self.canvas, 663, 390)  self.items[self.orange2.item] = self.orange2  self.hud = None  self.canvas.focus\_set()  self.canvas.bind('<Left>',  lambda \_: self.orange2.move(-10))  self.canvas.bind('<Right>',  lambda \_: self.orange2.move(10)) |
| --- | --- |

Initially, I struggled to make the images of the characters show up and move with the arrow keys when an object of them was instantiated (Test 1 and 2). In order to save time, I decided to start coding the battle with the characters as blocks instead of images as shown below and in the code above (line 06 and 21), so that I could focus on developing the game instead.



When I came back to making the characters show up as images, I realised that my error was that I was using Tkinter’s Label widget which I had used previously to make images show up. However, I learned that these labels could not be moved using key inputs. Therefore, to rectify this problem, I used Tkinter’s create\_image method, in order to display the image at the specified coordinates on the canvas, when the character object was instantiated, as shown below in line 01. This allowed me to then bind the keys to the object and made the character’s image move on the canvas with them (lines 02-04).

| 01  02  03  04 | self.orange = Orange(self.canvas, 200, 380, "oFight.png")  self.master.bind("<w>", self.orange.jump)  self.master.bind("<a>", self.orange.move\_left)  self.master.bind("<d>", self.orange.move\_right) |
| --- | --- |

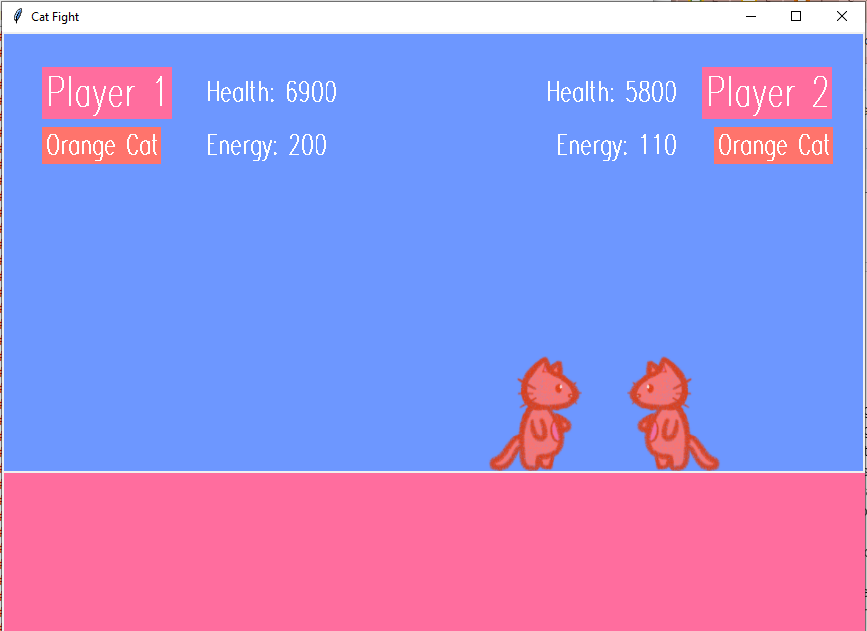
The next thing I needed to do was bind the different key inputs to the characters’ attacks, and make sure that these attacks would decrease the opponent’s health by a certain amount and, if needed, decrease the character’s amount of energy for the attack and display this amount on the screen (Test 4, 5, 6, 7). To do this, I made the damage1 attribute for player 1 and the damage2 attribute for player 2 public attributes so that in their individual takeDamage methods, their health would decrease by the amount inflicted by the opponent, as shown below (line 03). Each character has individual methods for each of their attacks, for example a normalAtk method, as well as a method to calculate their energy calcEnergy. Each attack method has individual integers stored in the damage1/damage2 attribute, which is different for each character.

| 01  02  03  04  05 | def takeDamage(self, damage2):  if self.health1 >= damage2:  self.health1 -= damage2  else:  self.health1 = 0 |
| --- | --- |

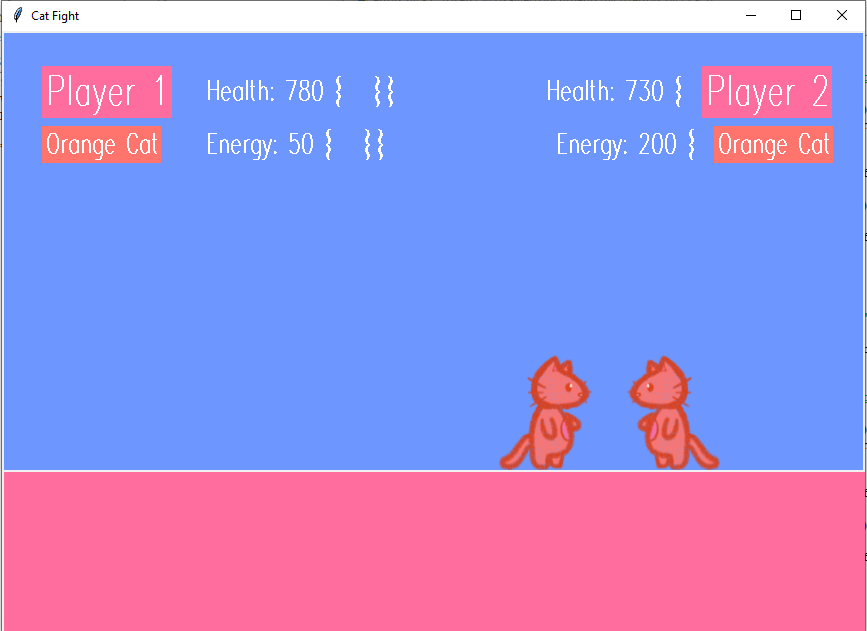
I also made sure that the damage they take wouldn’t decrease their health to a number less than 0, by making them take damage only when their health is greater than or equal to the damage inflicted (line 02 above), otherwise their health would equal 0 (line 05 above). I also had to make sure that the characters had enough energy to perform their special or ultimate attacks by using an if statement so that they only use these attacks if their energy is greater than or equal to the energy required as shown below in line 02, otherwise the attack would not be carried out and the opponent would not take damage.

| 01  02  03  04  05  06 | def specialAtk(self):  if self.energy1 >= 50:  self.energy1 -= 50  damage1 = 70  if selection2 == 1:  Orange2.takeDamage(self, damage1) |
| --- | --- |

To display the correct amount of energy and health on the battle screen, I used the Label widget to create a new label displaying what is stored in the health or energy attributes every time the values changed. This brought a problem of the labels overlapping each other as shown below, which not only made it seem like the wrong number was being displayed even if the right number was stored in the variable, but also made it very confusing for the players as the number on screen seemed incorrect.



As seen in the screenshot above, the numbers displayed are definitely incorrect as the maximum health is 1000, therefore the numbers that should be displayed are 690 and 580. To fix this, I decided to simply add a string of spaces after the text in each health or energy label, for example: lHealth = Label(self.canvas, text = ('Health:', self.health1, ' ')). This resulted in curly brackets being shown on screen as displayed below, but as this overall solved the problem, I decided to leave it as that in order to save time.



After that, I needed to make a timer. To do this, I imported the time module for Tkinter and set the time limit to ninety at first, and then used the Label widget to show the time remaining on the battle screen. Initially, I struggled to make the default time to be set to 90 seconds while also making sure that if the player were to choose a different time limit, that this would be implemented instead. I had a problem where the timer would start at 90 seconds no matter what was stored in the timeLimit attribute.

In the old code below, I first made an if statement which checked if the timeLimit attribute had the string ‘infinite’ stored (line 01), which would be what the user chose. Then, I had another if statement within, which set the time limit to 90 if the timeLimit attribute was not equal to 60 or 120 (line 03), otherwise the time limit would be set to 60 (line 08) or 120 (line 12) respectively.

In the new code, I made the same if statement to check if timeLimit was not equal to ‘infinite’ (line 01) and then used a different if statement within, which instead checked if timeLimit was equal to 60 or 120 (line 02) and then set the time limit to what was stored in the attribute timeLimit (line 03), otherwise the time limit was set 90 (line 08). I also made an else statement for the outer if statement, which displays a different label on the battle screen if the timeLimit attribute is set to ‘infinite’, as there would be no timer (line 17).

Old code:

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | if timeLimit != 'infinite':  if timeLimit != 60 or timeLimit != 120:  self.time\_limit = 90  self.start\_time = None  self.remaining\_time = self.time\_limit  self.start\_timer()  elif timeLimit == 60:  self.time\_limit = 60  self.start\_time = None  self.remaining\_time = self.time\_limit  self.start\_timer()  elif timeLimit == 120:  self.time\_limit = 120  self.start\_time = None  self.remaining\_time = self.time\_limit  self.start\_timer() |
| --- | --- |

New code:

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19 | if timeLimit != 'infinite':  if timeLimit == 60 or timeLimit == 120:  self.time\_limit = timeLimit  self.start\_time = None  self.remaining\_time = self.time\_limit  self.start\_timer()  else:  self.time\_limit = 90  self.start\_time = None  self.remaining\_time = self.time\_limit  self.start\_timer()  self.timer\_label = Label(self.master, text="",  font=('Modern 60 bold'), bg='#FF6D9E', fg='white')  self.timer\_label.place(x=410, y=40)  self.update\_timer()  else:  self.timer\_label = Label(self.master, text="∞",  font=('Modern 60 bold'), bg='#FF6D9E', fg='white')  self.timer\_label.place(x=410, y=40) |
| --- | --- |

However, I then encountered this error when no button was pressed in the settings to choose a time limit.



To fix this, I made the public attribute timeLimit equal to 90 in the constructor method of the Game class, which also set the default time limit to 90 seconds. The time limit could then be chosen by the player in the settings.

To make the pause button, I added a pause\_characters method to the Game class, where I unbinded all of the key inputs so that they could not be used (Test 12). The resume\_characters method was then used to bind all of the keys back onto the characters when the resume button is pressed. For the pause button overall, I made a new frame which appears on screen when the pause button is pressed, which would disappear after the player presses resume, using Tkinter’s place\_forget method.

Initially, I struggled with how to pause the timer on screen, and then make sure that it would resume from where the timer stopped. I used two methods within the Game class: pause\_timer and resume\_timer. I tried to use a get method to get the time the timer was paused at (line 03), and then use this to resume the timer at that time (line 12). However I realised that I could just use the attribute self.paused and make the timer either true or false in order to pause or resume the time. Therefore, I fixed this issue by changing the methods below:

Old code:

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21 | def pause\_timer(self):  self.timer\_paused = True  self.stopped\_time = int(self.remaining\_time.get())  self.paused\_time = Label(self.master,  text=self.stopped\_time, font=('Modern 60 bold'),  bg='#FF6D9E', fg='white')  self.paused\_time.place(x=410, y=40)  def resume\_timer(self):  if self.timer\_paused:  self.paused\_time.place\_forget()  self.time\_limit = self.stopped\_time  self.start\_time = None  self.start\_timer()  self.timer\_label = Label(self.master, text="",  font=('Modern 60 bold'), bg='#FF6D9E', fg='white')  self.timer\_label.place(x=410, y=40)  self.update\_timer()  self.timer\_paused = False |
| --- | --- |

New code:

| 01  02  03  04  05 | def pause\_timer(self):  self.paused = True  def resume\_timer(self):  self.paused = False |
| --- | --- |

I then added a ‘Retry’ button, ‘Back to character selection’ button and a ‘Back to starting page’ button onto the pause frame. At first, I set the command for the ‘Retry’ button to call a method called retry which would simply hide the pause frame and set self.paused to False in order to unpause the game, and then call the battle method again (Test 13). However, in doing this, the timer would overlap the old timer and go down quicker, and the characters would be unable to move. In order to fix this, I instead used the retry method to reset everything, including the health and energy attributes of the characters and the labels to display their health and energy. I then reset the timer as shown below by first checking if the time limit was not set to ‘infinite’ in the settings (line 01). This is so that the method wouldn’t try to reset a time limit that didn’t exist. I then stopped the current timer by checking if it was running, in which the self.start\_time attribute would not be set to None (line 02). I then used Tkinter’s after\_cancel function to stop the timer from updating after 1000 milliseconds (1 second) (line 03-04). The timer is then reset by setting the start time to None again (line 05) and then restarted with the start\_timer method (line 07).

| 01  02  03  04  05  06  07 | if timeLimit != 'infinite':  if self.start\_time is not None:  self.master.after\_cancel(self.master.after(1000,  self.update\_timer))  self.start\_time = None  self.remaining\_time = self.time\_limit  self.start\_timer() |
| --- | --- |

As I was unsure about how to go about the ‘Back to character selection’ button and ‘Back to starting page’ button, I decided to go back and make the rounds work instead, in order to save time (Test 10). To do this, I created a start\_new\_round method in the Game class, which is called when a character’s health is equal to 0 or when the timer ends. I then created a new attribute in the battle method called self.round\_no and set it to 1, and made a label to display the round number on the battle screen so that it is visible to the player. I then made it so that in the start\_new\_round method shown below, it would use an if statement in order to check if self.round\_no is equal to rounds (line 02), which is where the amount of rounds the user chose or the default amount of rounds if they didn’t choose, and if so, would call the end\_battle method (line 03) and end the current battle, which would ensure a new round wouldn’t start if unneeded. Otherwise, in the else statement, the self.round\_no attribute is incremented by 1 (line 26) and this is displayed on the battle screen using the label self.round\_label (line 27). Similarly to the retry method, I reset everything: the health and energy attributes of the characters (lines 05-08), the labels to display their health and energy (lines 09-24) and the timer (lines 38-44). I also added into the retry method a reset to the round number, where self.round\_no is set to 1 again.

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  4142  43  44 | def start\_new\_round(self):  if self.round\_no == rounds:  self.end\_battle()  else:  self.health1 = 1000  self.energy1 = 500  self.health2 = 1000  self.energy2 = 500  lHealth = Label(self.canvas, text = ('Health:',  self.health1, ' '), bg='#6D97FF', fg='white',  font=('Modern 20 bold'))  lHealth.place(x=200, y=42)  lEnergy = Label(self.canvas, text = ('Energy:',  self.energy1, ' '), bg='#6D97FF', fg='white',  font=('Modern 20 bold'))  lEnergy.place(x=200, y=95)  lHealth = Label(self.canvas, text = ('Health:',  self.health2, ' '), bg='#6D97FF', fg='white',  font=('Modern 20 bold'))  lHealth.place(x=540, y=42)  lEnergy = Label(self.canvas, text = ('Energy:',  self.energy2, ' '), bg='#6D97FF', fg='white',  font=('Modern 20 bold'))  lEnergy.place(x=550, y=95)    self.round\_no += 1  self.round\_label.config(text=f"Round  {self.round\_no}")    self.start\_label = Label(self.canvas, text="",  font=('Modern 70 bold'), bg='#FF6D9E', fg='white')  self.start\_label.place(x=260, y=200)  self.start\_label.config(text=f"Round  {self.round\_no} !")  self.start\_label.after(2000,  self.start\_label.place\_forget)  if timeLimit != 'infinite':  if self.start\_time is not None:  self.canvas.after\_cancel(self.canvas.  after(1000, self.update\_timer))  self.start\_time = None  self.remaining\_time = self.time\_limit  self.start\_timer() |
| --- | --- |

I then decided to go back to the actual battle between the characters. When I was initially struggling to make the character images show up and used coloured blocks in their place, I was also struggling with making it so that the characters would only attack within a certain distance of each other, and with how to detect collision so that the characters would not move past each other. At first, I decided to start with the collision detection first, with the method check\_collision shown below, as I thought this would aid me in making the characters only attack within a certain distance. To do this, I tried to get the x coordinates of the character’s images (lines 02-03), and then made an if statement in order to check if these character coordinates were touching (lines 05-08) and then return True if so, so that I could make the characters stop moving past each other with another method later.

| 01  02  03  04  05  06  07  08  09  10  11 | def check\_collision(self):  orange\_coords=self.canvas.coords(self.orange.imageID)  white2\_coords=self.canvas.coords(self.white2.imageID)  if (orange\_coords[2] > white2\_coords[0]and  orange\_coords[0] < white2\_coords[2]) or  (white2\_coords[2] > orange\_coords[0] and  white2\_coords[0] < orange\_coords[2]):  return True  else:  return False |
| --- | --- |

However, I was met with this error within the if statement..



Despite my attempts to fix this, I constantly ended up with this error and decided against making the characters stop moving when they touched, as not only was this time consuming, it is also uncommon in battle games and I no longer thought it was necessary.

I then decided to focus on making it so that the characters could only attack within a certain distance of each other. To do this, I imported Tkinter’s math module and added to each character’s attack methods the attribute distance, which stores the Euclidean distance between the two character’s coordinates on screen (lines 04-07 below), first checking which opponent is on screen using an if statement (line 03). I used if statements in each attack method so that the opponent would only take damage if they were within a certain distance of the attacker (line 08-09), otherwise nothing would happen. I decided to make the distances smaller for character attacks which dealt more damage, and bigger for character attacks that dealt less damage, in order to make the battle more fun and last longer. This can be seen below in Orange Cat’s normalAtk method when they try to attack Orange2, which would be the opponent.

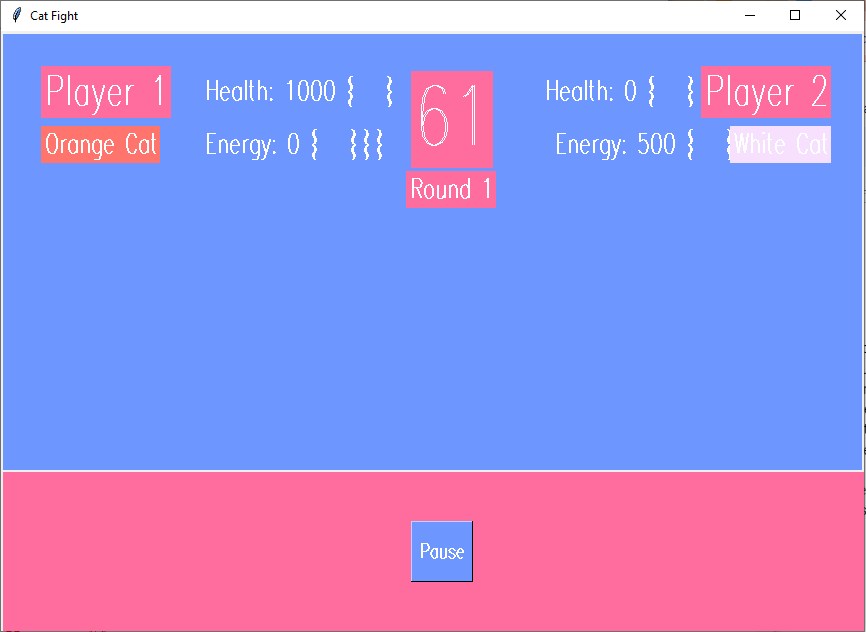
| 01  02  03  04  05  06  07  08  09 | def normalAtk(self):  damage1 = 20  if selection2 == 1:  distance = math.sqrt((self.orange.getLeft() -  self.orange2.getRight()) \*\* 2 +  (self.orange.getTop() - self.orange2.getTop()) \*\*  2)  if distance <= 425:  Orange2.takeDamage(self, damage1) |
| --- | --- |

I decided to deal with the ‘Back to character selection’ button and ‘Back to starting page’ button on the pause frame after I dealt with the after battle screen, as similar buttons needed to be used on both, therefore the pause frame buttons will be detailed in the ‘Create the after battle screen’ section below.

##### 4) Create the after battle screen

I made an after battle screen to display the winner, as well as to give three different options for the user to choose from: a ‘Retry’ button, a ‘Back to character selection’ button and a ‘Back to starting page’ button. To do this, I created an end\_battle method, which is called when either a character’s health is equal to 0 on the final round of the battle, the time has run out on the final round of the battle, or when the round number is equal to the amount of rounds the player has chosen, so that another round wouldn’t start and the battle would end.

Initially, I struggled to actually make the after battle screen show up, as the items in the battle screen wouldn’t disappear, as shown below.



Here, I tried to use Tkinter’s delete function by using the line self.canvas.delete("all") in order to delete everything on the canvas, however only the characters would disappear, as everything else was placed onto the root window self.master, while only the characters were placed on the battle canvas. I then tried to use Tkinter’s destroy function (line 01) and then made a new frame upon the root window self.master, as shown below (line 02).

| 01  02  03 | self.canvas.destroy()  self.end = Frame(self.master, bg='#68ffa9')  self.end.pack(fill="both", expand=True) |
| --- | --- |

However, this ended up only getting rid of the battle screen background as shown below, and revealed to me that the problem was that I had been using a frame for each screen which filled the previous screen, however for the battle, I had simply made a new canvas onto the root window.



This problem also interfered with the buttons on the after battle screen, because, as initially the screens such as the character selection screen had only been made to fill the starting page and not the battle screen, the button would then not be able to run the character selection method properly because the wrong screen was on display. This was also a problem with the ‘Back to character selection’ button and ‘Back to starting page’ button on the pause frame in the battle method.

In order to fix this, I first placed the battle canvas onto the player 2 character selection (the previous page which leads to the battle), rather than the root window self.master (line 02), and then placed the items that were on the battle screen onto the battle canvas self.canvas, rather than the root window, as shown below (line 06).

Old code:

| 01  02  03  04  05  06  07  08 | def battle(self):  self.canvas = Canvas(self.master, bg='#6D97FF',  width=863, height=600)  self.canvas.pack()  rectangle = Canvas(self.master, width=863, height=402,  bg='#FF6D9E', bd=0)  rectangle.place(x=0, y=439) |
| --- | --- |

New code:

| 01  02  03  04  05  06  07  08 | def battle(self):  self.canvas = Canvas(player2, bg='#6D97FF', width=863,  height=600)  self.canvas.pack()  rectangle = Canvas(self.canvas, width=863, height=402,  bg='#FF6D9E', bd=0)  rectangle.place(x=0, y=439) |
| --- | --- |

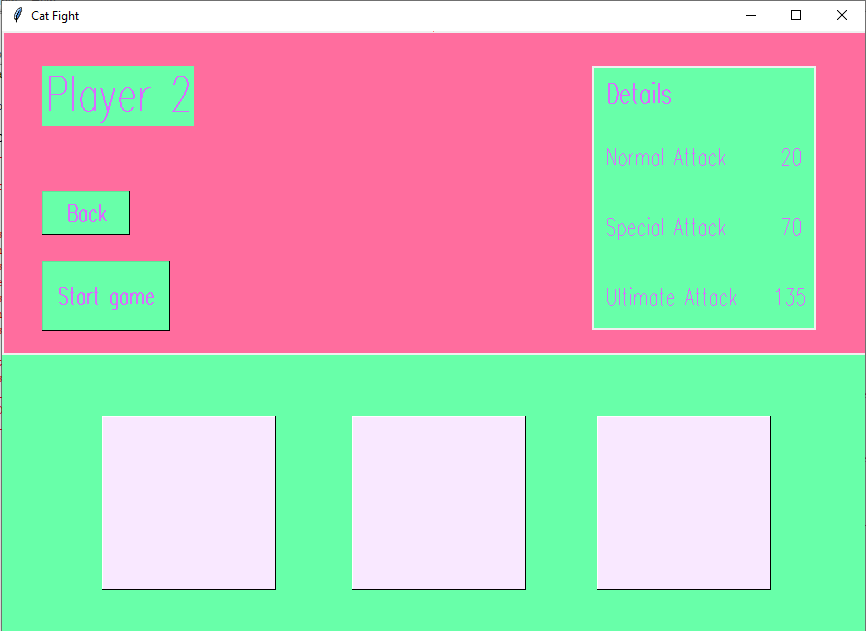
This ensured that the battle screen could be closed using Tkinter’s destroy function with the line self.canvas.destroy(), along with every item placed on it, as opposed to the entire window closing when the destroy function was used with the line self.master.destroy(). This was useful in getting the buttons on the after battle screen to work, which I will describe later on.

In order to make the after battle screen show up, I created a frame which was placed onto the battle canvas self.canvas with the lines below:

self.end = Frame(self.canvas, bg='#FF746C')

self.end.pack(fill="both", expand=True)

However, this was the result:



Instead of making a new screen with the background colour #FF746C, it would simply take the player back to the previous page of the battle: the character selection. I therefore added another frame on top of the player 2 character selection above with the lines below, which then displayed the after battle screen correctly.

self.end\_screen = Frame(player2, bg='#FF746C')

self.end\_screen.pack(fill="both", expand=True)

In order to display the winner on the after battle screen, I created an empty list called winners in the constructor method of the Game class, which either “Player 1”, “Player 2” or “Draw” was added to the list depending on the outcome of each round. I then used the count method to count the amount of wins each player had by counting how many times “Player 1” or “Player 2” appeared in the list, and stored them in their respective attributes player1\_wins and player2\_wins. I then used the if statement below to compare the amount of wins of each player (lines 01 and 07), to store an overall winner in the attribute self.actualWinner (lines 02 and 08). The winner is then displayed on the after battle screen using the label winner\_label, or a draw is displayed otherwise.

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  1617  18 | if player1\_wins > player2\_wins:  self.actualWinner = "Player 1"  winner\_label = Label(self.end\_screen, text="Player  1 wins!", font=('Modern 50 bold'), bg='#DA6BFF',  fg='white')  winner\_label.place(x=20, y=20)  elif player1\_wins < player2\_wins:  self.actualWinner = "Player 2"  winner\_label = Label(self.end\_screen, text="Player  2 wins!", font=('Modern 50 bold'), bg='#DA6BFF',  fg='white')  winner\_label.place(x=20, y=20)  else:  self.actualWinner = "Draw"  winner\_label = Label(self.end\_screen,  text="Draw!", font=('Modern 50 bold'),  bg='#DA6BFF', fg='white')  winner\_label.place(x=20, y=20) |
| --- | --- |

I then used the attribute self.actualWinner in an if statement as shown below (line 01) in order to display the winning (lines 03-07) and losing images (lines 09-13) of the correct characters on the end battle screen.

| 01  02  03  04  05  06  07  08  09  10  11  12  13 | if self.actualWinner == "Player 1":  if selection1 == 1:  winner = PhotoImage(file="oWin.png")  winner.image = winner  winner\_label = Label(self.end\_screen,  image=winner, bg='#FF746C',bd=0)  winner\_label.place(x=10, y=130)  if selection2 == 1:  loser = PhotoImage(file="oLose.png")  loser.image = loser  loser\_label = Label(self.end\_screen,  image=loser, bg='#FF746C',bd=0)  loser\_label.place(x=300, y=200) |
| --- | --- |

After that, I decided to focus on the ‘Back to character selection’ and ‘Back to starting page’ buttons. First, I focussed on the ‘Back to character selection’ button on the after battle screen (Test 14). Initially, I created a button with the command self.charaSelect1, which I assumed would go back to the character selection by calling the charaSelect1 method. However, as described earlier, the character selection screen had only been made to fill the starting page therefore the button would then not be able to run the character selection method properly because the wrong screen was on display (the after battle screen). Instead, as I had moved everything on the battle screen from the root window to the battle canvas as described earlier also, I decided to make a new method to be called with the button, called end\_chara, shown below. This method uses the destroy function to remove the end screen (line 06), the battle screen (line 07), and the character selection for Player 2 (line 08), essentially bringing the player back to the character selection for Player 1 in this way. I also reset the character selections of Player 1 and Player 2 to 0 (lines 05-06) to ensure a choice is made again when it is validated.

| 01  02  03  04  05  06  07  08 | def end\_chara(self):  global selection1  global selection2  selection1 = 0  selection2 = 0  self.end\_screen.destroy()  self.canvas.destroy()  player2.destroy() |
| --- | --- |

I dealt with the ‘Back to starting page’ button in the same way, creating a new method called end\_back, which also used the destroy function to remove the character selection for Player 1, bringing the player back to the starting screen, as well as resetting the character selections (Test 15).

I then made a new function called end\_retry for the retry button on the after battle screen, which made a new battle canvas with the same name self.canvas, and placed everything from the battle canvas back on screen again. Although the method is almost identical to the battle method, I had to create this new method in order to reset the timer similarly to the retry method described earlier, as well as for the problem I ended up with below.

After retrying, the after battle screen from the previous battle would display underneath the new one as shown below (Test 13).



To fix this, I recognised it was because the after battle screen was being placed onto the wrong frame after the game was retried, and therefore I created a public attribute called check, which is set to 1 in the original battle method, and set to 2 in the end\_retry function. Then, I made an if statement as shown below which checked which screen the after battle screen should be placed on: the character selection screen for Player 2 if check stores 1 (line 02), or the previous after battle screen if check stores 2 (line 05).

| 01  02  03  04  05  06  07 | if check == 1:  self.end\_screen = Frame(player2, bg='#FF746C')  self.end\_screen.pack(fill="both", expand=True)  elif check == 2:  self.end\_screen = Frame(self.end\_screen,  bg='#FF746C')  self.end\_screen.pack(fill="both", expand=True) |
| --- | --- |

#### 

#### Testing to inform development

I will now test the program using my test plan on page 50 from the design section, using videos to evidence.

| **Test number** | **Function being tested** | **Method of input** | **Input** | **Expected result** | **Actual result** | **Video evidence** |
| --- | --- | --- | --- | --- | --- | --- |
| Test 1 | Character selection | Button clicked on the character selection screen | Orange Cat button clicked (selection(1 or 2) = 1) | The orange cat character will appear on the battle screen to be played by the user | As expected | <https://youtu.be/HS3H_nnm98E> |
| White Cat button clicked (selection = 2) | The white cat character will appear on the battle screen to be played by the user | As expected | <https://youtu.be/3J_tDjHZav0> |
| Black Cat button clicked (selection = 3) | The black cat character will appear on the battle screen to be played by the user | As expected | <https://youtu.be/S2KpNdMyJNM> |
| No button clicked (selection = 0) | The user will prompted with an alert message to select a character before moving onto the next screen | As expected | <https://youtu.be/PuzMeUbBgss> |
| Test 2 | Character movement | Pressing a button on the keyboard | A | Player 1 will move left | As expected | <https://youtu.be/MdWMGbn2j3Y> |
| D | Player 1 will move right | As expected | <https://youtu.be/MdWMGbn2j3Y> |
| W | Player 1 will jump | As expected | <https://youtu.be/av2O2PwSDzY> |
| S | Player 1 will guard (They will not be able to take damage) | As expected | <https://youtu.be/AfnZ6wFuXY0> |
| Left | Player 2 will move left | As expected | <https://youtu.be/iEWwo0uXJhc> |
| Right | Player 2 will move right | As expected | <https://youtu.be/iEWwo0uXJhc> |
| Up | Player 2 will jump | As expected | <https://youtu.be/av2O2PwSDzY> |
| Down | Player 2 will guard (They will not be able to take damage) | As expected | <https://youtu.be/cw69NPiSzTE> |
| Test 3 | Collision with borders | When in battle, head into a border of the game screen using the directional buttons on the keyboard | D (Player 1)/Right (Player 2) to direct the characters to the right border of the game screen | The characters will not be able to move past the borders of the game screen and will stop moving at the border. | The characters won’t move off-screen, however when approaching the left border, half of the character image is able to go off-screen, and when approaching the right border, the character will stop before the border is reached. | Player1: <https://youtu.be/MdWMGbn2j3Y>  Player2: <https://youtu.be/iEWwo0uXJhc> |
| A (Player 1)/Left (Player 2) to direct the characters to the left border of the game screen |
| N/A | Collision with opponent | When in battle, head into the direction of the opposing player using the directional buttons on the keyboard | Directional buttons on the keyboard used to direct the characters to their opponents | The characters will not be able to move past each other and will stop moving when the border of one character touches the border of the other | During development, I decided that this was no longer necessary. | N/A |
| Test 4 | Character using normal attack | Pressing a button on the keyboard | E (Player 1)/ B (Player 2) | Damage will be dealt to the opponent by the normal attack damage amount of the player and the opponent’s health will decrease by this much | As expected | <https://youtu.be/t2Txl0pFHws> |
| Test 5 | Character using special attack | Pressing a button on the keyboard | R (Player 1)/ N (Player 2) | The player’s energy will decrease by 50 and damage will be dealt to the opponent by the special attack damage amount of the player, decreasing the opponent’s health by this much | As expected | <https://youtu.be/K3mthgL1X5o> |
| Test 6 | Character using ultimate attack | Pressing a button on the keyboard | T (Player 1)/ M (Player 2) | The player’s energy will decrease by 100 and damage will be dealt to the opponent by the special attack damage amount of the player, decreasing the opponent’s health by this much | As expected | <https://youtu.be/vgUMMV-bedE> |
| Test 7 | Character recharging energy | Pressing a button on the keyboard | Q (Player 1)/ L (Player 2) | The player’s energy will increase by 10 if the energy bar is not full, otherwise their energy will remain at 500 | As expected | <https://youtu.be/8LxxmAN_now> |
| Test 8 | Displaying instructions | Clicking the ‘How to play’ button on the starting screen | N/A | Displays a new window with the instructions on the key inputs for both Player 1 and Player 2 | As expected | <https://youtu.be/aiR97YwNRIM> |
| Test 9 | Displaying settings | Clicking the ‘Settings’ button on the starting screen | N/A | Displays a new window with the options for the amount of rounds and the time limit in battle | As expected | <https://youtu.be/W38I6cH1Sto> |
| Test 10 | Selecting amount of rounds | Button clicked in settings | ‘1’ button clicked (rounds = 1) | One round will be played during battle | As expected | <https://youtu.be/QucjGBnDeG4> |
| ‘3’ button clicked (rounds = 3) | Three rounds will be played during battle, and the player who wins the most rounds will win | As expected | <https://youtu.be/ymluSPd-3kU> |
| ‘5’ button clicked (rounds = 5) | Five rounds will be played during battle, and the player who wins the most rounds will win | As expected | <https://youtu.be/Mq0-xUX9iV0> |
| No button clicked | The rounds will be set to a default of three rounds | As expected | <https://youtu.be/UHXkDeBZrbI> |
| Test 11 | Selecting the time limit of the rounds | Button clicked in settings | ‘60’ button clicked | Each round will have timer of 60 seconds | As expected | <https://youtu.be/XZ1ohb_xs-E> |
| ‘90’ button clicked | Each round will have timer of 90 seconds | As expected | <https://youtu.be/Bs3E3j-a-U4> |
| ‘120’ button clicked | Each round will have timer of 120 seconds | As expected | <https://youtu.be/LyFYMWDbpVs> |
| ‘∞’ button clicked | Each round will continue for an infinite amount of time and will only end when a character is defeated | As expected | <https://youtu.be/RvmC3-Fqmwg> |
| No button clicked | The default time limit will be set to 90 seconds | As expected | <https://youtu.be/UHXkDeBZrbI> |
| Test 12 | Selecting the pause button | Button clicked during battle | N/A | Pauses the game so that characters are unable to move or attack and displays a window with options to retry, return to character selection or return to the starting page, and can cross off the window to resume | As expected, except ‘cross off to resume’ was replaced with a simple ‘Resume’ button for ease of use | <https://youtu.be/hZYa_j8o1Z8> |
| Test 13 | Selecting the ‘Retry’ button | Button clicked after pausing or after battle | N/A | Restarts the battle from the start with full energy and health bars | As expected | <https://youtu.be/ekNx2Abpj1Q> |
| Test 14 | Selecting the ‘Character Selection’ button | Button clicked after pausing or after battle | N/A | Ends the current battle and returns to the character selection | Sometimes works as expected, however sometimes the buttons do not work and the player has to use the back button to return to the character selection from the starting page in order to select a character. | <https://youtu.be/CRufwAwLY7w> |
| Test 15 | Selecting the ‘Starting Page’ button | Button clicked after pausing or after battle | N/A | Ends the current battle and returns to the starting page | As expected | <https://youtu.be/yc4wdHEzFBI> |

By testing the program according to my test plan, I have shown that most of the features are functional. The unexpected results in test 3 do not greatly affect the gameplay, therefore, due to time constraints, I will leave this to be considered in the project’s evaluation. However, for test 14, I managed to fix this simply by altering the end\_chara and battle\_chara methods which lead back to the character selection from the after battle screen and the battle screen respectively, by, instead of just going back to the character selection for player 1 using the destroy method as shown below (old code, lines 08 and 16), I used the destroy methods to go back to the starting page (new code, lines 09 and 19), and then called the character selection method again (new code, lines 10 and 20). This then ensures that the buttons are clickable, and also doesn’t display a previously selected character on screen either.

Old code:

| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | def end\_chara(self):  global selection1  global selection2  selection1 = 0  selection2 = 0  self.end\_screen.destroy()  self.canvas.destroy()  player2.destroy()  def battle\_chara(self):  global selection1  global selection2  selection1 = 0  selection2 = 0  self.canvas.destroy()  player2.destroy() |
| --- | --- |

New code:

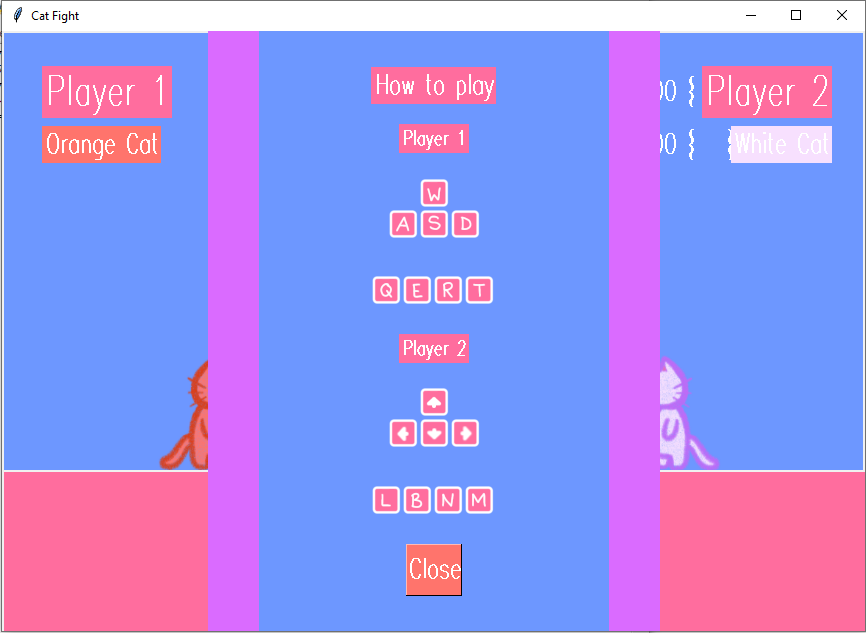
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20 | def end\_chara(self):  global selection1  global selection2  selection1 = 0  selection2 = 0  self.end\_screen.destroy()  self.canvas.destroy()  player2.destroy()  start.destroy()  self.charaSelect1()  def battle\_chara(self):  global selection1  global selection2  selection1 = 0  selection2 = 0  self.canvas.destroy()  player2.destroy()  start.destroy()  self.charaSelect1() |
| --- | --- |

After testing with potential users (GCSE and A Level students), key problems that they pointed out to me were:

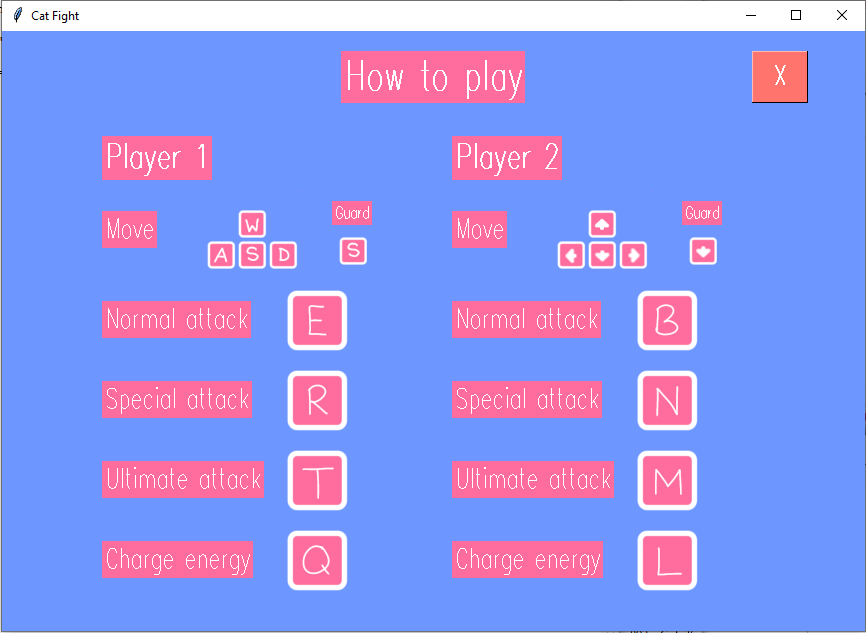
* After starting battle, it is easy to forget how to play as there is no reminder of the instructions
  + Players were also unaware that they could guard
* There is no indication of the winner after each round
* After each round, the characters’ positions do not reset

I will detail how I fixed these below.

For the first problem, I originally planned to have the key inputs displayed at the bottom of the battle screen as detailed in the design section (page 49). However, in attempting to implement this, I found that the images of the keys would disappear when two or more keys were pressed simultaneously. I therefore decided to implement the instructions in battle in a different way: by adding a button in the pause frame called ‘How to play’ which details the instructions again, as shown below.



In the main ‘How to play’ button on the starting screen, I also added an extra instruction for the keys to guard, as shown below.



To solve the second problem, initially, I added in a label which disappears after two seconds to display the winner on screen, before ending the battle or starting the next round in the start\_new\_round method, with the code below, for example.

| 01  02  03  04 | self.win = Label(self.canvas, text="Player 2 wins!",  font=('Modern 70 bold'), bg='#FF6D9E', fg='white')  self.win.place(x=200, y=200)  self.win.after(2000, self.win.place\_forget) |
| --- | --- |

However, due to the after method in line 04 being used multiple times in my start\_new\_round method (i.e. to display the round number at the start of the round and to update the timer), the rounds began to add up abnormally. Therefore, instead, I decided to display a new label that appears throughout the battle through all of the rounds, which indicates the winner of the last round, through the use of the self.winners list, which stores the winner of each round. I added this label shown below where each character’s health is equal to 0, or where the time has run out, which leads to either ‘Player 1’, ‘Player 2’ or ‘Draw’ being stored in the self.winners list.

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white')

To solve the third problem, in the start\_new\_round method, I ‘reset’ the character positions with the code below, by first deleting the current character off the screen (line 02), and then instantiating a new object of the character at the correct coordinates (line 03) and then binding the keys to the new object instead, shown partially below (lines 05-11).

| 01  02  03  04  05  06  07  08  09  10  11 | if selection1 == 1:  self.orange.canvas.delete(self.orange.imageID)  self.orange = Orange(self.canvas, 200, 380,  "oFight.png")  self.master.bind("<w>", self.orange.jump)  self.master.bind("<a>", self.orange.move\_left)  self.master.bind("<d>",self.orange.move\_right)  self.master.bind("<KeyPress-s>",  self.orange.guard)  self.master.bind("<KeyRelease-s>",  self.orange.unguard) |
| --- | --- |

### 

### Evaluation

#### Testing to inform evaluation

In order to clarify the satisfaction of my intended users (GCSE and A level students), I have created a questionnaire to see if users find my program user-friendly and suitable to them. The questionnaire, and the frequency of each users’ responses after testing the finished solution are shown below.

##### User questionnaire

| **Question** | **Answer (Y/N)** | **Comment** |
| --- | --- | --- |
| Were the instructions clear? | Y (8)  N (0) | None |
| Were you easily able to tell what to do on each screen (e.g. starting screen, character selection screen)? | Y (8)  N (0) | None |
| Was the gameplay fun and able to help you to destress? | Y (8)  N (0) | None |
| Are you satisfied with the overall look of the game? | Y (8)  N (0) | None |
| Was the battle easy to understand and play? | Y (8)  N (0) | None |
| Were the characters easy to control? | Y (8)  N (0) | Both players cannot move at the same time. |
| These are the problems outlined by users that I sought to resolve:   * After starting battle, it is easy to forget how to play as there is no reminder of the instructions   + Players were also unaware that they could guard * There is no indication of the winner after each round * After each round, the characters’ positions do not reset   Do you believe that these problems have been fully resolved? | Y (8)  N (0) | None |
| Are there any other problems that you believe still have to be fixed? | Y (0)  N (8) | None |

In this questionnaire, it can be seen that users’ responses to the completed game are positive. The users are satisfied with the resolved problems, and agree that the game meets the design objectives. As there is a problem highlighted about the movement of both players, I will discuss this in the maintenance and development section on page 99.

#### Success of the solution

I will now compare my success criteria in the analysis section on page 13 to the finished program, detailing whether or not these criteria were met with test evidence.

| **Criteria** | **How to evidence** | **Met or not met** | **Screenshot reference** |
| --- | --- | --- | --- |
| Main starting page with start button, instructions on how to play, settings | Screenshot of the main starting page that shows where the ‘How to play’ and settings button are | Met | 1 |
| Engaging and vibrant design | Screenshots of the windows with bright colours and appealing fonts, characters and world design | Met | 2 |
| Visible pause button during battle with pause symbol | Screenshot of the battle screen with a pause button visible and not drowned out by the background | Met | 3 |
| Settings menu on a separate page | Screenshot of the settings menu that is separate from the starting page, showing the ability to change the amount of rounds and the time limit in a game | Met | 4 |
| Instructions on a separate page, shows which keys are used for player 1 and player 2 | Screenshot of the instructions of how to play separate from the starting page, detailing the movement (jump, left, right, guard), normal attack, special attack, ultimate attack buttons for both player one and player two | Met | 5 |
| Character selection on a separate page | Screenshot of the character selection with the option to play against a computer or against another player in character selection, button to press to change the second player (opponent) into player two, default is against a computer | Partially met - Due to time constraints, I did not add in an option to play against a computer, and I will detail this in the maintenance and development section on page 99. | 6 |

| **Screenshot reference** | **Screenshots** |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

#### Describe the final product

After having completed the game, I can conclude that most of the design objectives that were detailed in the design section on page 46 were met. The objectives 1c, 4d and 5f below were partially met, however in a different way than intended, which I will detail below to discuss each objective and explain why each of them were successfully met with evidence through the testing of the finished solution.

| **Requirement** | **Met or not met** | **Explanation of why the requirement was met/not met** | **Evidence** |
| --- | --- | --- | --- |
| 1a) The game will start with a vibrant and enticing starting page as preferred by potential users, with three different buttons: a settings button, which opens a new window with buttons to change the amount of rounds in a battle and buttons to change how long the rounds will be; a ‘How to Play’ button, which opens a new window and displays in a simple format the key inputs needed for both Player 1 and Player 2 so that players know how to play; and a start button, which leads to the character selection for Player 1 for quick and easy use. | Met | This requirement was met as the starting screen indeed did use vibrant colours, as well as having a settings button which opens a new frame to change the rounds and time limit, a ‘How to play’ button opening a new frame which displays the instructions simply, and a start button which leads to the character selection for Player 1. | <https://youtu.be/T65GVzqiXZo> |
| 1b) The character selection will have a selection of three different characters, displaying their different attacks and their damage amounts in a box so users have more choice in what character they would like to play. This would then lead to the character selection for Player 2, displayed similarly, and then this would lead into the battle. | Met | There were indeed three characters to choose from, all with their individual damage amounts displayed upon selecting them, as well as a next button to lead to the character selection for Player 2, and a back button to go back to the previous page. This is similar for the character selection for Player 2, however with a ‘Start’ button which leads into the battle. | <https://youtu.be/Jfz0-gMIcUI> |
| 1c) The battle will display the characters selected by the players, as well as their health and energy bars at the top of the screen and the timer and the rounds remaining between each players’ bars so that players are aware of how much time, health and energy they have remaining and can strategise their next moves from there.  When the time runs out or a player is defeated, it will lead to the after battle page, where the players will have three different options: retry, return to character selection and return to starting page, which gives them the choice of whether they would like to play again with the same characters or settings or change them. | Partially met | The battle does indeed display the characters selected by the players, as well as a timer with either the player’s choice of time limit or the default, and the current round as chosen by the player or the default. However, instead of health and energy bars, instead I displayed the remaining health and energy with labels in a numeric form initially for testing to ensure the values stored in the attributes were changing correctly. However, due to time constraints, I was unable to figure out how to convert these bars to visually represent the remaining health and energy. When the time runs out or a player is defeated at the end of all the rounds, the players are indeed led to an after battle page with the options to retry, return to character selection or return to starting page. | Selected characters: Test 1 (page 73)  Time limit: Test 11 (page 79)  Rounds: Test 10 (page 79)  Health labels: Test 4/5/6 (pages 77-78)  Energy labels: Test 7 (page 78)  After battle screen: Test 13/14/15 (pages 80-81) |
| 2a) The screen size will be moderately sized in order to fit most screens (863x600 pixels). | Met | The screen size was, as planned, 863x600 pixels, which was the size of my starting screen image. | This screen size can be seen throughout all testing videos and screenshots. |
| 2b) There will be three different characters: an orange cat, a black cat and a white cat to give players a choice of what character they would like to play as well as make sure characters don’t get mixed up on the battle screen. | Met | There were indeed three different characters: an orange, black and white cat. Players can select the same character, however, as Player 1 faces right and Player 2 faces left, the characters do not get mixed up. | Test 1 (page 75) |
| 2c) Each page within the game will follow a similar colour scheme of blue, orange, pink, purple and green for a vibrant design that was preferred by potential users. | Met | This colour scheme was indeed followed throughout. | This colour scheme can be seen throughout all testing videos and screenshots. |
| 2d) In battle, Player 1 will be facing right and Player 2 will be facing left so that players can easily tell who is who. | Met | Player 1 was indeed facing right and Player 2 facing left in battle. | This can be seen in multiple testing videos e.g. Test 2, Test 3 (pages 75-76) |
| 2e) In battle, the characters will be approximately 100x120 pixels to make sure they are not too big or too small on the screen size. | Met | The characters were approximately 100x120 pixels, though I had planned to make them a little bigger in order to fit the screen better, but due to time constraints, I left them as is. | This can be seen in multiple testing videos e.g. Test 2, Test 3 (pages 75-76) |
| 2f) When moving, the characters will move 10 pixels at a time so that characters can move quickly across the screen to attack or evade attacks. | Met | The characters indeed did move 10 pixels at a time when the keys are used, which ensures they aren’t moving too fast or too slow. | Test 3 (page 76) |
| 3a) The player will be able to choose how many rounds they would like to play to give them more choice to play in a way that will help them to destress. | Met | The player was able to choose how many rounds they wanted to play. | Test 10 (page 79) |
| 3b) The player will be able to choose how much time they would like the round(s) to be to give them more choice to play in a way that will help them to destress. | Met | The player was able to choose the time limit of each round. | Test 11 (page 79) |
| 3c) The players will be able to choose which character they would like to play as to give players a choice of what character they would like to play as well as make sure characters don’t get mixed up on the battle screen. | Met | The players were indeed able to choose which character they would like to play as in the character selection. | Test 1 (page 75) |
| 3d) The players will be able to control the movement of the characters left, right and up using the arrow keys (Player 2) and WASD keys (Player 1) as these are keys that are commonly used to move characters. | Met | The players were indeed able to control the movement in this way. | Test 2 (page 75) |
| 3e) The players will be able to use key inputs to perform three different attacks on their opponents: normal attack, special attack and ultimate attack, as well as to recharge their energy and have the ability to guard to give a more enticing battle experience that lasts longer. | Met | The players were indeed able to use key inputs to perform these different attacks, guard and recharge their energy. | Guard: Test 2 (page 75)  Normal attack: Test 4 (page 77)  Special attack: Test 5 (page 77)  Ultimate attack: Test 6 (page 78)  Recharge energy: Test 7 (page 78) |
| 3f) The player will be able to view a ‘How to Play’ screen that opens in a new window by clicking a button that is on the initial starting screen so that players can easily access the instructions. | Met | The ‘How to play’ button on the starting screen does open a new frame which displays the instructions simply. | <https://youtu.be/T65GVzqiXZo> |
| 3g) The player will be able to view and change the settings that opens in a new window by clicking a button that is on the initial starting screen so that players can easily change their settings before starting the battle. | Met | The settings button on the starting screen does open a new frame to change the rounds and the time limit. | <https://youtu.be/T65GVzqiXZo> |
| 3h)The player will be able to go back to the initial starting screen from the character selection screen and will be able to go back to the initial starting screen, character selection screen or battle after a battle has ended so that players can change their characters or settings if they would like to. | Met | The player was able to go back to the initial starting screen from the character selection through the back buttons, and was able to go back to the starting page, character selection or battle (through the retry button) on the after battle page. | <https://youtu.be/U-cPtOgjutw> |
| 3i) The players will be able to pause the battle, and through this, will be able to retry, go back to the character selection or go back to the starting page, which gives players the choice of what they would like to do rather than just having a resume button. | Met | The players were indeed able to do this through the pause button in battle. | <https://youtu.be/cPcT_u3qdJw> |
| 4a) When a character is attacked, their health will go down by the amount of damage dealt by the attack. | Met | This can be seen to be true as the health labels change when attacked. | Test 4/5/6 (pages 77-78) |
| 4b) When a character uses a special attack or an ultimate attack, their energy will go down by a certain amount. | Met | This can be seen to be true as the energy labels change when attacking. | Test 5/6 (pages 77-78) |
| 4c) When a player uses a key input to recharge their energy, their energy will go up by 10. | Met | The players were indeed able to recharge their energy using a key input, as evidenced in the testing section. | Test 7 (page 78) |
| 4d) The characters must not be able to go off-screen to ensure that the battle only happens on-screen. | Partially met | The characters were able to go partially off-screen when moving left but stopped moving before going fully off-screen; and to the right border, the characters stopped moving just before the border of the screen. | Test 3 (page 76) |
| 4e) The characters should only be able to attack within a certain distance of their opponent, which ensures that players cannot continuously keep attacking each other. | Met | The characters were indeed only able to perform different attacks within a certain distance of their opponent. This can be seen in the video where Orange Cat attempts to perform special or ultimate attacks, draining their energy, but does not drain White Cat’s health as they are too far away. | <https://youtu.be/efb0DtcN1as> |
| 4f) The battle will end when a character is defeated (their health is equal to 0), when time runs out, or when the players manually end it through the pause button options. | Met | The battle did indeed end when a character was defeated or when the time ran out, leading to the after battle page after all rounds had been completed. The players were indeed able to manually end the battle through the pause button, the next screen depending on which button they pressed. | After battle page: <https://youtu.be/QA-I2ZUkbpQ>  Pause button: <https://youtu.be/cPcT_u3qdJw> |
| 4g) The winner of each round must be recorded in order to declare a final winner by the end of all of the rounds. | Met | The winner of each round was indeed recorded in a list as detailed in the testing section (page 84), in order to declare the final winner on the after battle screen by counting how many times they appear in the list. This list can be seen on screen when a player wins a round. | <https://youtu.be/QIxJQFFi7qk> |
| 5a) The current round will be displayed. | Met | The current round was indeed displayed during battle. | <https://youtu.be/QIxJQFFi7qk> |
| 5b) The time remaining will be displayed. | Met | The time remaining was indeed displayed during battle. | This can be seen in multiple videos displaying the battle, for example: <https://youtu.be/QIxJQFFi7qk> |
| 5c) The amount of health each player has remaining will be displayed. | Met | The amount of health remaining was indeed displayed during battle. | Test 4/5/6 (pages 77-78) |
| 5d) The amount of energy each player has will be displayed. | Met | The amount of energy remaining was indeed displayed during battle. | Test 7 (page 78) |
| 5e) Which characters are playing that have been chosen by the players will be displayed. | Met | The characters chosen by the players were indeed displayed during battle. | Test 1 (page 75) |
| 5f) Instructions on how to play the game will be displayed during battle and also through a button on the starting screen. | Partially met | The instructions were indeed displayed through a ‘How to play’ button in the starting screen, however, in battle, another similar button was used instead in the pause frame to display the instructions again, as shown in the testing section. | <https://youtu.be/dO5S7j5a9_0> |

#### Maintenance and development

I shall now list the good and bad points of the completed program and note its limitations, and how these may be resolved in the future.

##### Good points:

* As concluded from the user questionnaire on page 85, the game can be seen as user-friendly and a way to destress
* The look of the completed game closely matches my initial design, and gained a positive response from players
* The instructions on how to play were deemed clear by the players in the questionnaire
* The battle was easy enough to play to help students relax and have fun
* All buttons and key inputs work as desired
* The program mostly functions as desired, with the exception of a few points which I will describe below

##### Bad points:

* The players are unable to move simultaneously, which makes the battle more difficult than intended
* There is no option to play against a computer as planned in my initial success criteria

##### 

##### Limitations and how they may be resolved:

The issue of the characters being unable to move simultaneously was a problem I did attempt to fix by attempting to change how Tkinter processes the events, by, instead of binding the keys directly to the moving methods, using the KeyPress and KeyRelease functions which I used for the guarding methods instead. However, this had no effect, therefore, due to time constraints, I was unable to solve this problem, as it may have required me to rewrite the methods of each character in order to let them move simultaneously.

Being able to play against a computer which would have allowed just a single player to play by themselves was again something I didn’t implement due to time constraints, however this could be considered a possible extension that could be implemented in a future version of the game.

Another possible extension to be considered is the ability to resize the game window. If the user does expand the window, the frames in the background can be seen, which was unintended, therefore this could be improved in the future.

The problem of the characters only moving just past the border of the screen or just before it is also an issue I did not resolve due to time constraints, as the overall main aim (for the characters to not move off-screen) was met. In future versions, this could be improved so that the characters remain strictly within the borders of the screen, from one edge to another.

Another extension could be the use of animations, so that it is easier for the player to understand what is happening on screen, as well as making the game more enticing. Once again, this was not implemented due to time constraints.

### Bibliography

Tkinter tutorial documentation: <https://tk-tutorial.readthedocs.io/_/downloads/en/latest/pdf/>

Timer: <https://www.w3resource.com/python-exercises/tkinter/python-tkinter-events-and-event-handling-exercise-10.php>

Display images: <https://stackoverflow.com/questions/31959379/how-to-insert-a-photo-into-a-tkinter-window-no-pil-of-pillow>

Make character images move with arrow keys: <https://www.tutorialspoint.com/how-to-move-an-image-in-tkinter-canvas-with-arrow-keys>

Pause and unpause timer: <https://pyseek.com/2021/11/countdown-timer-in-python/>

### Appendix

Code:

from tkinter import \*

from tkinter import messagebox

import time

import math

class Orange: #Create a class for the character Orange Cat

def \_\_init\_\_(self, canvas, x, y, imagePath):

global damage1 #Public attribute to be used in attack methods

global guarding1 #Public attribute to check if the character is guarding

guarding1 = False #Set to false initially as character isn't guarding

self.canvas = canvas

self.image = PhotoImage(file=imagePath)

self.imageID = self.canvas.create\_image(200, 380, image=self.image)

self.imageWidth = self.image.width()

self.imageHeight = self.image.height()

def move\_left(self, event):

if self.getLeft() > 0: #Make sure left side of character does not go past the left of the screen

self.canvas.move(self.imageID, -10, 0) #Move the character image left

def move\_right(self, event):

if self.getRight() < self.canvas.winfo\_width(): #Make sure right side of character does not go past the right of the screen

self.canvas.move(self.imageID, 10, 0) #Move the character image right

def getLeft(self):

return self.canvas.coords(self.imageID)[0] #Get the coordinates of the left side of the character image

def getTop(self):

return self.canvas.coords(self.imageID)[1] #Get the coordinates of the top side of the character image

def getRight(self):

return self.canvas.coords(self.imageID)[0] + self.imageWidth #Get the coordinates of the right side of the character image

def getBottom(self):

return self.canvas.coords(self.imageID)[1] + self.imageHeight #Get the coordinates of the bottom side of the character image

def jump(self, event):

current\_y = self.canvas.coords(self.imageID)[1] #Store the current y coordinates of the character image

jumpDistance = 50 #Store how many pixels for the character to jump

self.canvas.move(self.imageID, 0, -jumpDistance) #Make the character move up by the value stored in jumpDistance

self.canvas.after(200, lambda: self.canvas.move(self.imageID, 0, jumpDistance)) #Return the character back to its original position after 200ms

def guard(self, event):

global guarding1 #Public attribute to check if the character is guarding

guarding1 = True #Make the character guard

def unguard(self, event):

global guarding1

guarding1 = False #Stop the character from guarding

def calcEnergy(self):

if self.energy1 != 500:

self.energy1 += 10

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Display new value of self.energy1 on screen

lEnergy.place(x=200, y=95)

else:

self.energy1 == 500 #Make sure value stored in self.energy1 doesn't go above 500

def normalAtk(self):

damage1 = 20

if selection2 == 1:

distance = math.sqrt((self.orange.getLeft() - self.orange2.getRight()) \*\* 2 + (self.orange.getTop() - self.orange2.getTop()) \*\* 2) #Calculate distance between player 1 and player 2

if distance <= 425: #Check if distance is close enough to attack

Orange2.takeDamage(self, damage1) #Opponent takes damage if so

elif selection2 == 2:

distance = math.sqrt((self.orange.getLeft() - self.white2.getRight()) \*\* 2 + (self.orange.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 425:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.orange.getLeft() - self.black2.getRight()) \*\* 2 + (self.orange.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 425:

Black2.takeDamage(self, damage1)

def specialAtk(self):

if self.energy1 >= 50: #Check if energy is sufficient to be able to perform attack

self.energy1 -= 50 #Deplete energy regardless if close enough to attack

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Update new value of self.energy1 on screen

lEnergy.place(x=200, y=95)

damage1 = 70

if selection2 == 1:

distance = math.sqrt((self.orange.getLeft() - self.orange2.getRight()) \*\* 2 + (self.orange.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 300:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.orange.getLeft() - self.white2.getRight()) \*\* 2 + (self.orange.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 300:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.orange.getLeft() - self.black2.getRight()) \*\* 2 + (self.orange.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 300:

Black2.takeDamage(self, damage1)

def ultAtk(self):

if self.energy1 >= 100:

self.energy1 -= 100

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

damage1 = 135

if selection2 == 1:

distance = math.sqrt((self.orange.getLeft() - self.orange2.getRight()) \*\* 2 + (self.orange.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 215:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.orange.getLeft() - self.white2.getRight()) \*\* 2 + (self.orange.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 215:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.orange.getLeft() - self.black2.getRight()) \*\* 2 + (self.orange.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 215:

Black2.takeDamage(self, damage1)

def takeDamage(self, damage2):

if guarding1 == False: #Check if character isn't guarding

if self.health1 >= damage2: #Check if character has enough health to take the full damage

self.health1 -= damage2

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Update health on screen

lHealth.place(x=200, y=42)

else:

self.health1 = 0 #Make sure health doesn't go below 0

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Update health on screen

lHealth.place(x=200, y=42)

self.winners.append("Player 2") #Add Player 2 to the list winners as Player 1 has lost (health=0)

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white') #Display the list on screen to show who won the last round

lose.place(x=320, y=180)

self.start\_new\_round()

class Orange2:

def \_\_init\_\_(self, canvas, x, y, imagePath):

global damage2

global guarding2

guarding2 = False

self.canvas = canvas

self.image = PhotoImage(file=imagePath)

self.imageID = self.canvas.create\_image(675, 380, image=self.image)

self.imageWidth = self.image.width()

self.imageHeight = self.image.height()

def move\_left(self, event):

if self.getLeft() > 0:

self.canvas.move(self.imageID, -10, 0)

def move\_right(self, event):

if self.getRight() < self.canvas.winfo\_width():

self.canvas.move(self.imageID, 10, 0)

def getLeft(self):

return self.canvas.coords(self.imageID)[0]

def getTop(self):

return self.canvas.coords(self.imageID)[1]

def getRight(self):

return self.canvas.coords(self.imageID)[0] + self.imageWidth

def getBottom(self):

return self.canvas.coords(self.imageID)[1] + self.imageHeight

def jump(self, event):

current\_y = self.canvas.coords(self.imageID)[1]

jumpDistance = 50

self.canvas.move(self.imageID, 0, -jumpDistance)

self.canvas.after(500, lambda: self.canvas.move(self.imageID, 0, jumpDistance))

def guard(self, event):

global guarding2

guarding2 = True

def unguard(self, event):

global guarding2

guarding2 = False

def calcEnergy(self):

if self.energy2 != 500:

self.energy2 += 10

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=712, y=95)

else:

self.energy2 == 500

def normalAtk(self):

damage2 = 20

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.orange2.getRight()) \*\* 2 + (self.orange.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 425:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.orange2.getRight()) \*\* 2 + (self.white.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 425:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.orange2.getRight()) \*\* 2 + (self.black.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 425:

Black.takeDamage(self, damage2)

def specialAtk(self):

if self.energy2 >= 50:

self.energy2 -= 50

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=712, y=95)

damage2 = 70

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.orange2.getRight()) \*\* 2 + (self.orange.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 300:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.orange2.getRight()) \*\* 2 + (self.white.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 300:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.orange2.getRight()) \*\* 2 + (self.black.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 300:

Black.takeDamage(self, damage2)

def ultAtk(self):

if self.energy2 >= 100:

self.energy2 -= 100

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=712, y=95)

damage2 = 135

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.orange2.getRight()) \*\* 2 + (self.orange.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 215:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.orange2.getRight()) \*\* 2 + (self.white.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 215:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.orange2.getRight()) \*\* 2 + (self.black.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 215:

Black.takeDamage(self, damage2)

def takeDamage(self, damage1):

if guarding2 == False:

if self.health2 >= damage1:

self.health2 -= damage1

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=712, y=95)

else:

self.health2 = 0

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=712, y=95)

self.winners.append("Player 1")

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white')

lose.place(x=320, y=180)

self.start\_new\_round()

class White:

def \_\_init\_\_(self, canvas, x, y, imagePath):

global damage1

global guarding1

guarding1 = False

self.canvas = canvas

self.image = PhotoImage(file=imagePath)

self.imageID = self.canvas.create\_image(200, 380, image=self.image)

self.imageWidth = self.image.width()

self.imageHeight = self.image.height()

def move\_left(self, event):

if self.getLeft() > 0:

self.canvas.move(self.imageID, -10, 0)

def move\_right(self, event):

if self.getRight() < self.canvas.winfo\_width():

self.canvas.move(self.imageID, 10, 0)

def getLeft(self):

return self.canvas.coords(self.imageID)[0]

def getTop(self):

return self.canvas.coords(self.imageID)[1]

def getRight(self):

return self.canvas.coords(self.imageID)[0] + self.imageWidth

def getBottom(self):

return self.canvas.coords(self.imageID)[1] + self.imageHeight

def jump(self, event):

current\_y = self.canvas.coords(self.imageID)[1]

jumpDistance = 50

self.canvas.move(self.imageID, 0, -jumpDistance)

self.canvas.after(500, lambda: self.canvas.move(self.imageID, 0, jumpDistance))

def guard(self, event):

global guarding1

guarding1 = True

def unguard(self, event):

global guarding1

guarding1 = False

def calcEnergy(self):

if self.energy1 != 500:

self.energy1 += 10

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

else:

self.energy1 == 500

def normalAtk(self):

damage1 = 35

if selection2 == 1:

distance = math.sqrt((self.white.getLeft() - self.orange2.getRight()) \*\* 2 + (self.white.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 400:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.white.getLeft() - self.white2.getRight()) \*\* 2 + (self.white.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 400:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.white.getLeft() - self.black2.getRight()) \*\* 2 + (self.white.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 400:

Black2.takeDamage(self, damage1)

def specialAtk(self):

if self.energy1 >= 50:

self.energy1 -= 50

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

damage1 = 60

if selection2 == 1:

distance = math.sqrt((self.white.getLeft() - self.orange2.getRight()) \*\* 2 + (self.white.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 310:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.white.getLeft() - self.white2.getRight()) \*\* 2 + (self.white.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 310:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.white.getLeft() - self.black2.getRight()) \*\* 2 + (self.white.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 310:

Black2.takeDamage(self, damage1)

def ultAtk(self):

if self.energy1 >= 100:

self.energy1 -= 100

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

damage1 = 130

if selection2 == 1:

distance = math.sqrt((self.white.getLeft() - self.orange2.getRight()) \*\* 2 + (self.white.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 230:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.white.getLeft() - self.white2.getRight()) \*\* 2 + (self.white.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 230:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.white.getLeft() - self.black2.getRight()) \*\* 2 + (self.white.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 230:

Black2.takeDamage(self, damage1)

def takeDamage(self, damage2):

if guarding1 == False:

if self.health1 >= damage2:

self.health1 -= damage2

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=200, y=42)

else:

self.health1 = 0

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=200, y=42)

self.winners.append("Player 2")

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white')

lose.place(x=320, y=180)

self.start\_new\_round()

class White2:

def \_\_init\_\_(self, canvas, x, y, imagePath):

global damage2

global guarding2

guarding2 = False

self.canvas = canvas

self.image = PhotoImage(file=imagePath)

self.imageID = self.canvas.create\_image(675, 380, image=self.image)

self.imageWidth = self.image.width()

self.imageHeight = self.image.height()

def move\_left(self, event):

if self.getLeft() > 0:

self.canvas.move(self.imageID, -10, 0)

def move\_right(self, event):

if self.getRight() < self.canvas.winfo\_width():

self.canvas.move(self.imageID, 10, 0)

def getLeft(self):

return self.canvas.coords(self.imageID)[0]

def getTop(self):

return self.canvas.coords(self.imageID)[1]

def getRight(self):

return self.canvas.coords(self.imageID)[0] + self.imageWidth

def getBottom(self):

return self.canvas.coords(self.imageID)[1] + self.imageHeight

def jump(self, event):

current\_y = self.canvas.coords(self.imageID)[1]

jumpDistance = 50

self.canvas.move(self.imageID, 0, -jumpDistance)

self.canvas.after(500, lambda: self.canvas.move(self.imageID, 0, jumpDistance))

def guard(self, event):

global guarding2

guarding2 = True

def unguard(self, event):

global guarding2

guarding2 = False

def calcEnergy(self):

if self.energy2 != 500:

self.energy2 += 10

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=729, y=95)

else:

self.energy2 == 500

def normalAtk(self):

damage2 = 35

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.white2.getRight()) \*\* 2 + (self.orange.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 400:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.white2.getRight()) \*\* 2 + (self.white.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 400:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.white2.getRight()) \*\* 2 + (self.black.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 400:

Black.takeDamage(self, damage2)

def specialAtk(self):

if self.energy2 >= 50:

self.energy2 -= 50

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=729, y=95)

damage2 = 60

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.white2.getRight()) \*\* 2 + (self.orange.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 310:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.white2.getRight()) \*\* 2 + (self.white.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 310:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.white2.getRight()) \*\* 2 + (self.black.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 310:

Black.takeDamage(self, damage2)

def ultAtk(self):

if self.energy2 >= 100:

self.energy2 -= 100

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=729, y=95)

damage2 = 130

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.white2.getRight()) \*\* 2 + (self.orange.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 230:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.white2.getRight()) \*\* 2 + (self.white.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 230:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.white2.getRight()) \*\* 2 + (self.black.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 230:

Black.takeDamage(self, damage2)

def takeDamage(self, damage1):

if guarding2 == False:

if self.health2 >= damage1:

self.health2 -= damage1

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=729, y=95)

else:

self.health2 = 0

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=729, y=95)

self.winners.append("Player 1")

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white')

lose.place(x=320, y=180)

self.start\_new\_round()

class Black:

def \_\_init\_\_(self, canvas, x, y, imagePath):

global damage1

global guarding1

guarding1 = False

self.canvas = canvas

self.image = PhotoImage(file=imagePath)

self.imageID = self.canvas.create\_image(200, 380, image=self.image)

self.imageWidth = self.image.width()

self.imageHeight = self.image.height()

def move\_left(self, event):

if self.getLeft() > 0:

self.canvas.move(self.imageID, -10, 0)

def move\_right(self, event):

if self.getRight() < self.canvas.winfo\_width():

self.canvas.move(self.imageID, 10, 0)

def getLeft(self):

return self.canvas.coords(self.imageID)[0]

def getTop(self):

return self.canvas.coords(self.imageID)[1]

def getRight(self):

return self.canvas.coords(self.imageID)[0] + self.imageWidth

def getBottom(self):

return self.canvas.coords(self.imageID)[1] + self.imageHeight

def jump(self, event):

current\_y = self.canvas.coords(self.imageID)[1]

jumpDistance = 50

self.canvas.move(self.imageID, 0, -jumpDistance)

self.canvas.after(500, lambda: self.canvas.move(self.imageID, 0, jumpDistance))

def guard(self, event):

global guarding1

guarding1 = True

def unguard(self, event):

global guarding1

guarding1 = False

def calcEnergy(self):

if self.energy1 != 500:

self.energy1 += 10

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

else:

self.energy1 == 500

def normalAtk(self):

damage1 = 25

if selection2 == 1:

distance = math.sqrt((self.black.getLeft() - self.orange2.getRight()) \*\* 2 + (self.black.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 410:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.black.getLeft() - self.white2.getRight()) \*\* 2 + (self.black.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 410:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.black.getLeft() - self.black2.getRight()) \*\* 2 + (self.black.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 410:

Black2.takeDamage(self, damage1)

def specialAtk(self):

if self.energy1 >= 50:

self.energy1 -= 50

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

damage1 = 50

if selection2 == 1:

distance = math.sqrt((self.black.getLeft() - self.orange2.getRight()) \*\* 2 + (self.black.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 330:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.black.getLeft() - self.white2.getRight()) \*\* 2 + (self.black.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 330:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.black.getLeft() - self.black2.getRight()) \*\* 2 + (self.black.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 330:

Black2.takeDamage(self, damage1)

def ultAtk(self):

if self.energy1 >= 100:

self.energy1 -= 100

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

damage1 = 150

if selection2 == 1:

distance = math.sqrt((self.black.getLeft() - self.orange2.getRight()) \*\* 2 + (self.black.getTop() - self.orange2.getTop()) \*\* 2)

if distance <= 200:

Orange2.takeDamage(self, damage1)

elif selection2 == 2:

distance = math.sqrt((self.black.getLeft() - self.white2.getRight()) \*\* 2 + (self.black.getTop() - self.white2.getTop()) \*\* 2)

if distance <= 200:

White2.takeDamage(self, damage1)

elif selection2 == 3:

distance = math.sqrt((self.black.getLeft() - self.black2.getRight()) \*\* 2 + (self.black.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 200:

Black2.takeDamage(self, damage1)

def takeDamage(self, damage2):

if guarding1 == False:

if self.health1 >= damage2:

self.health1 -= damage2

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=200, y=42)

else:

self.health1 = 0

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=200, y=42)

self.winners.append("Player 2")

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white')

lose.place(x=320, y=180)

self.start\_new\_round()

class Black2:

def \_\_init\_\_(self, canvas, x, y, imagePath):

global damage2

global guarding2

guarding2 = False

self.canvas = canvas

self.image = PhotoImage(file=imagePath)

self.imageID = self.canvas.create\_image(675, 380, image=self.image)

self.imageWidth = self.image.width()

self.imageHeight = self.image.height()

def move\_left(self, event):

if self.getLeft() > 0:

self.canvas.move(self.imageID, -10, 0)

def move\_right(self, event):

if self.getRight() < self.canvas.winfo\_width():

self.canvas.move(self.imageID, 10, 0)

def getLeft(self):

return self.canvas.coords(self.imageID)[0]

def getTop(self):

return self.canvas.coords(self.imageID)[1]

def getRight(self):

return self.canvas.coords(self.imageID)[0] + self.imageWidth

def getBottom(self):

return self.canvas.coords(self.imageID)[1] + self.imageHeight

def jump(self, event):

current\_y = self.canvas.coords(self.imageID)[1]

jumpDistance = 50

self.canvas.move(self.imageID, 0, -jumpDistance)

self.canvas.after(500, lambda: self.canvas.move(self.imageID, 0, jumpDistance))

def guard(self, event):

global guarding2

guarding2 = True

def unguard(self, event):

global guarding2

guarding2 = False

def calcEnergy(self):

if self.energy2 != 500:

self.energy2 += 10

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=729, y=95)

else:

self.energy2 == 500

def normalAtk(self):

damage2 = 25

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.black2.getRight()) \*\* 2 + (self.orange.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 400:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.black2.getRight()) \*\* 2 + (self.white.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 400:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.black2.getRight()) \*\* 2 + (self.black.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 400:

Black.takeDamage(self, damage2)

def specialAtk(self):

if self.energy2 >= 50:

self.energy2 -= 50

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=729, y=95)

damage2 = 50

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.black2.getRight()) \*\* 2 + (self.orange.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 300:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.black2.getRight()) \*\* 2 + (self.white.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 300:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.black2.getRight()) \*\* 2 + (self.black.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 300:

Black.takeDamage(self, damage2)

def ultAtk(self):

if self.energy2 >= 100:

self.energy2 -= 100

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=729, y=95)

damage2 = 150

if selection1 == 1:

distance = math.sqrt((self.orange.getLeft() - self.black2.getRight()) \*\* 2 + (self.orange.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 200:

Orange.takeDamage(self, damage2)

elif selection1 == 2:

distance = math.sqrt((self.white.getLeft() - self.black2.getRight()) \*\* 2 + (self.white.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 200:

White.takeDamage(self, damage2)

elif selection1 == 3:

distance = math.sqrt((self.black.getLeft() - self.black2.getRight()) \*\* 2 + (self.black.getTop() - self.black2.getTop()) \*\* 2)

if distance <= 200:

Black.takeDamage(self, damage2)

def takeDamage(self, damage1):

if guarding2 == False:

if self.health2 >= damage1:

self.health2 -= damage1

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=729, y=95)

else:

self.health2 = 0

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=729, y=95)

self.winners.append("Player 1")

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white')

lose.place(x=320, y=180)

self.start\_new\_round()

class Game(Frame):

def \_\_init\_\_(self, master):

super().\_\_init\_\_(master)

self.master = master

self.master.title("Cat Fight")

self.master.geometry("863x600")

self.bg = PhotoImage(file="starting.png") #Keep a reference to the PhotoImage object

label1 = Label(self.master, image=self.bg) #Display the image as the background of the window

label1.place(x=0, y=0)

global timeLimit #Public attributes to be used in other methods

global rounds

timeLimit = 90 #Make 90 the default value

rounds = 3 #Make 3 the default value

global selection1

global selection2

selection1 = 0 #Make 0 the default value

selection2 = 0

self.winners = [] #List to store the winner of each round

self.actualWinner = None #Attribute to store the end winner

settings = Button(self.master, text='Settings', fg='#DA6BFF', bg='#6BFFA9', bd=1, font=('Modern 17 bold'),

command=self.options, height=2, width=8)

settings.place(x=25, y=25)

instructions = Button(self.master, text='How to play', fg='#DA6BFF', bg='#6BFFA9', bd=1, font=('Modern 17 bold'),

command=self.howToPlay, height=1, width=15)

instructions.place(x=140, y=40)

start = Button(self.master, text='Start', fg='#FF746C', bg='#6BFFA9', bd=5, font=('Modern 35 bold'),

command=self.charaSelect1, height=1, width=8)

start.place(x=350, y=470)

def howToPlay(self):

inst = Frame(root, bg='#6D97FF')

inst.pack(fill="both", expand="yes")

top = Label(inst, text = "How to play", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top.pack(pady=20)

left = Label(inst, text="Player 1", bg='#FF6D9E', fg='white', font=('Modern 25 bold'))

left.place(x=100, y=105)

right = Label(inst, text="Player 2", bg='#FF6D9E', fg='white', font=('Modern 25 bold'))

right.place(x=450, y=105)

p1Move = Label(inst, text="Move", bg='#FF6D9E', fg='white', font=('Modern 20'))

p1Move.place(x=100, y=180)

p2Move = Label(inst, text="Move", bg='#FF6D9E', fg='white', font=('Modern 20'))

p2Move.place(x=450, y=180)

p1Guard = Label(inst, text="Guard", bg='#FF6D9E', fg='white', font=('Modern 12'))

p1Guard.place(x=330, y=170)

p2Guard = Label(inst, text="Guard", bg='#FF6D9E', fg='white', font=('Modern 12'))

p2Guard.place(x=680, y=170)

p1Normal = Label(inst, text="Normal attack", bg='#FF6D9E', fg='white', font=('Modern 20'))

p1Normal.place(x=100, y=270)

p2Normal = Label(inst, text="Normal attack", bg='#FF6D9E', fg='white', font=('Modern 20'))

p2Normal.place(x=450, y=270)

p1Special = Label(inst, text="Special attack", bg='#FF6D9E', fg='white', font=('Modern 20'))

p1Special.place(x=100, y=350)

p2Special = Label(inst, text="Special attack", bg='#FF6D9E', fg='white', font=('Modern 20'))

p2Special.place(x=450, y=350)

p1Ult = Label(inst, text="Ultimate attack", bg='#FF6D9E', fg='white', font=('Modern 20'))

p1Ult.place(x=100, y=430)

p2Ult = Label(inst, text="Ultimate attack", bg='#FF6D9E', fg='white', font=('Modern 20'))

p2Ult.place(x=450, y=430)

p1Charge = Label(inst, text="Charge energy", bg='#FF6D9E', fg='white', font=('Modern 20'))

p1Charge.place(x=100, y=510)

p2Charge = Label(inst, text="Charge energy", bg='#FF6D9E', fg='white', font=('Modern 20'))

p2Charge.place(x=450, y=510)

close = Button(inst, text='X', fg='white', bg='#FF746C', bd=1, font=('Modern 20 bold'), #Close the frame when the button is clicked

command = inst.pack\_forget, height=1, width=4)

close.place(x=750, y=20)

wasd = PhotoImage(file="wasd.png")

wasd.image = wasd

wasd\_label = Label(inst, image=wasd, bd=0)

wasd\_label.place(x=200, y=160)

arrows = PhotoImage(file="arrows.png")

arrows.image = arrows

arrows\_label = Label(inst, image=arrows, bd=0)

arrows\_label.place(x=550, y=160)

s = PhotoImage(file="s.png")

s.image = s

s\_label = Label(inst, image=s, bg='#6D97FF', bd=0)

s\_label.place(x=335, y=205)

down = PhotoImage(file="down.png")

down.image = down

down\_label = Label(inst, image=down, bg='#6D97FF', bd=0)

down\_label.place(x=685, y=205)

e = PhotoImage(file="e.png")

e.image = e

e\_label = Label(inst, image=e, bd=0)

e\_label.place(x=265, y=240)

b = PhotoImage(file="b.png")

b.image = b

b\_label = Label(inst, image=b, bd=0)

b\_label.place(x=615, y=240)

r = PhotoImage(file="r.png")

r.image = r

r\_label = Label(inst, image=r, bd=0)

r\_label.place(x=265, y=320)

n = PhotoImage(file="n.png")

n.image = n

n\_label = Label(inst, image=n, bd=0)

n\_label.place(x=615, y=320)

t = PhotoImage(file="t.png")

t.image = t

t\_label = Label(inst, image=t, bd=0)

t\_label.place(x=265, y=400)

m = PhotoImage(file="m.png")

m.image = m

m\_label = Label(inst, image=m, bd=0)

m\_label.place(x=615, y=400)

q = PhotoImage(file="q.png")

q.image = q

q\_label = Label(inst, image=q, bd=0)

q\_label.place(x=265, y=480)

l = PhotoImage(file="l.png")

l.image = l

l\_label = Label(inst, image=l, bd=0)

l\_label.place(x=615, y=480)

def options(self):

sett = Frame(root, bg='#6D97FF')

sett.pack(fill="both", expand="yes")

top = Label(sett, text = "Settings", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top.pack(pady=20)

roundsTitle = Label(sett, text="Rounds", bg='#FF6D9E', fg='white', font=('Modern 25 bold'))

roundsTitle.pack(pady=45)

rounds1 = Button(sett, text='1', fg='white', bg='#FF6D9E', bd=1, font=('Modern 15 bold'),

command = self.oneRound, height=2, width=8)

rounds1.place(x=170,y=220)

rounds3 = Button(sett, text='3', fg='white', bg='#FF6D9E', bd=1, font=('Modern 15 bold'),

command = self.threeRound, height=2, width=8)

rounds3.place(x=390, y=220)

rounds5 = Button(sett, text='5', fg='white', bg='#FF6D9E', bd=1, font=('Modern 15 bold'),

command = self.fiveRound, height=2, width=8)

rounds5.place(x=600,y=220)

timeLimit = Label(sett, text="Time limit", bg='#FF6D9E', fg='white', font=('Modern 25 bold'))

timeLimit.pack(pady=120)

time60 = Button(sett, text='60s', fg='white', bg='#FF6D9E', bd=1, font=('Modern 15 bold'),

command = self.sixty, height=2, width=8)

time60.place(x=160,y=440)

time90 = Button(sett, text='90s', fg='white', bg='#FF6D9E', bd=1, font=('Modern 15 bold'),

command = self.ninety, height=2, width=8)

time90.place(x=310,y=440)

time120 = Button(sett, text='120s', fg='white', bg='#FF6D9E', bd=1, font=('Modern 15 bold'),

command = self.one20, height=2, width=8)

time120.place(x=460,y=440)

timeInf = Button(sett, text='∞', fg='white', bg='#FF6D9E', bd=1, font=('Modern 15 bold'),

command = self.infinite, height=2, width=8)

timeInf.place(x=610,y=440)

close = Button(sett, text='X', fg='white', bg='#FF746C', bd=1, font=('Modern 20 bold'), #Close the frame with the button is clicked

command = sett.pack\_forget, height=1, width=4)

close.place(x=750, y=20)

def oneRound(self):

global rounds #Public attribute to be used in other methods

rounds = 1 #Store player's choice of rounds clicked in buttons in options method

def threeRound(self):

global rounds

rounds = 3

def fiveRound(self):

global rounds

rounds = 5

def sixty(self):

global timeLimit #Public attribute to be used in other methods

timeLimit = 60 #Store player's choice of time limit clicked in buttons in options method

def ninety(self):

global timeLimit

timeLimit = 90

def one20(self):

global timeLimit

timeLimit = 120

def infinite(self):

global timeLimit

timeLimit = 'infinite'

def charaSelect1(self):

global start

start = Frame(root, bg='#68ffa9')

start.pack(fill="both", expand="yes")

wCat = PhotoImage(file="white2.png")

wCat.image = wCat #Store reference to image

wCat\_label = Label(start, image=wCat, bd=0) #Create label to display image

white = Button(start, image=wCat, bg='#f9e8ff', bd=1, font=('Modern 17 bold'),

command = self.wChar1, height=170, width=170)

white.place(x=350,y=385)

oCat = PhotoImage(file="orange2.png")

oCat.image = oCat

oCat\_label = Label(start, image=oCat, bd=0)

orange = Button(start, image=oCat, bg='#f9e8ff', bd=1, font=('Modern 17 bold'),

command = self.oChar1, height=170, width=170)

orange.place(x=100,y=385)

bCat = PhotoImage(file="black2.png")

bCat.image = bCat

bCat\_label = Label(start, image=bCat, bd=0)

black = Button(start, image=bCat, bg='#f9e8ff', bd=1, font=('Modern 17 bold'),

command = self.bChar1, height=170, width=170)

black.place(x=595,y=385)

rectangle = Canvas(start, width=863, height=320, bg='#FF6D9E', bd=0)

rectangle.place(x=0, y=0)

info = Canvas(start, width=220, height=260, bg='#68ffa9', bd=0)

info.place(x=590, y=35)

top = Label(start, text = "Player 1", bg='#68ffa9', fg='#DA6BFF', font=('Modern 35 bold'))

top.place(x=40, y=35)

back = Button(start, text='Back', fg='#DA6BFF', bg='#68ffa9', bd=1, font=('Modern 17 bold'),

command = start.destroy, height=1, width=8)

back.place(x=40,y=160)

toP2 = Button(start, text='Next', fg='#DA6BFF', bg='#68ffa9', bd=1, font=('Modern 17 bold'),

command = self.charaSelect2, height=2, width=8)

toP2.place(x=40,y=230)

def charaSelect2(self):

if selection1 == 0:

messagebox.showwarning("Alert", "Must choose a character") #Make sure player cannot progress if a character hasn't been chosen

else:

global player2

player2 = Frame(start, bg='#68ffa9')

player2.pack(fill="both", expand="yes")

wCat = PhotoImage(file="white2.png")

wCat.image = wCat

wCat\_label = Label(player2, image=wCat, bd=0)

white = Button(player2, image=wCat, bg='#f9e8ff', bd=1, font=('Modern 17 bold'),

command = self.wChar2, height=170, width=170)

white.place(x=350,y=385)

oCat = PhotoImage(file="orange2.png")

oCat.image = oCat

oCat\_label = Label(player2, image=oCat, bd=0)

orange = Button(player2, image=oCat, bg='#f9e8ff', bd=1, font=('Modern 17 bold'),

command = self.oChar2, height=170, width=170)

orange.place(x=100,y=385)

bCat = PhotoImage(file="black2.png")

bCat.image = bCat

bCat\_label = Label(player2, image=bCat, bd=0)

black = Button(player2, image=bCat, bg='#f9e8ff', bd=1, font=('Modern 17 bold'),

command = self.bChar2, height=170, width=170)

black.place(x=595,y=385)

r = Canvas(player2, width=863, height=320, bg='#FF6D9E', bd=0)

r.place(x=0,y=0)

info = Canvas(player2, width=220, height=260, bg='#68ffa9', bd=0)

info.place(x=590, y=35)

top = Label(player2, text = "Player 2", bg='#68ffa9', fg='#DA6BFF', font=('Modern 35 bold'))

top.place(x=40, y=35)

back = Button(player2, text='Back', fg='#DA6BFF', bg='#68ffa9', bd=1, font=('Modern 17 bold'),

command = player2.destroy, height=1, width=8)

back.place(x=40,y=160)

toGame = Button(player2, text='Start game', fg='#DA6BFF', bg='#68ffa9', bd=1, font=('Modern 17 bold'),

command = self.battle, height=2, width=12)

toGame.place(x=40,y=230)

def oChar1(self):

global selection1 #Public attribute to be used in other methods to check which character has been chosen

selection1 = 1

photo = PhotoImage(file="oCS2.png")

photo.image = photo

photo\_label = Label(start, image=photo, bg='#FF6D9E',bd=0) #Display the chosen character on screen

photo\_label.place(x=260, y=30)

details = Label(start, text = "Details", bg='#68ffa9', fg='#DA6BFF', font=('Modern 20 bold')) #Display the character's details on screen

details.place(x=600, y=45)

nAtk = Label(start, text = "Normal Attack 20", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

nAtk.place(x=600, y=110)

sAtk = Label(start, text = "Special Attack 70", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

sAtk.place(x=600, y=180)

uAtk = Label(start, text = "Ultimate Attack 135", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

uAtk.place(x=600, y=250)

def oChar2(self):

global selection2

selection2 = 1

photo = PhotoImage(file="oCS2.png")

photo.image = photo

photo\_label = Label(player2, image=photo, bg='#FF6D9E',bd=0)

photo\_label.place(x=260, y=30)

details = Label(player2, text = "Details", bg='#68ffa9', fg='#DA6BFF', font=('Modern 20 bold'))

details.place(x=600, y=45)

nAtk = Label(player2, text = "Normal Attack 20", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

nAtk.place(x=600, y=110)

sAtk = Label(player2, text = "Special Attack 70", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

sAtk.place(x=600, y=180)

uAtk = Label(player2, text = "Ultimate Attack 135", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

uAtk.place(x=600, y=250)

def wChar1(self):

global selection1

selection1 = 2

photo = PhotoImage(file="wCS2.png")

photo.image = photo

photo\_label = Label(start, image=photo, bg='#FF6D9E',bd=0)

photo\_label.place(x=260, y=30)

details = Label(start, text = "Details", bg='#68ffa9', fg='#DA6BFF', font=('Modern 20 bold'))

details.place(x=600, y=45)

nAtk = Label(start, text = "Normal Attack 35", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

nAtk.place(x=600, y=110)

sAtk = Label(start, text = "Special Attack 60", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

sAtk.place(x=600, y=180)

uAtk = Label(start, text = "Ultimate Attack 130", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

uAtk.place(x=600, y=250)

def wChar2(self):

global selection2

selection2 = 2

photo = PhotoImage(file="wCS2.png")

photo.image = photo

photo\_label = Label(player2, image=photo, bg='#FF6D9E',bd=0)

photo\_label.place(x=260, y=30)

details = Label(player2, text = "Details", bg='#68ffa9', fg='#DA6BFF', font=('Modern 20 bold'))

details.place(x=600, y=45)

nAtk = Label(player2, text = "Normal Attack 35", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

nAtk.place(x=600, y=110)

sAtk = Label(player2, text = "Special Attack 60", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

sAtk.place(x=600, y=180)

uAtk = Label(player2, text = "Ultimate Attack 130", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

uAtk.place(x=600, y=250)

def bChar1(self):

global selection1

selection1 = 3

photo = PhotoImage(file="bCS2.png")

photo.image = photo

photo\_label = Label(start, image=photo, bg='#FF6D9E',bd=0)

photo\_label.place(x=260, y=30)

details = Label(start, text = "Details", bg='#68ffa9', fg='#DA6BFF', font=('Modern 20 bold'))

details.place(x=600, y=45)

nAtk = Label(start, text = "Normal Attack 25", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

nAtk.place(x=600, y=110)

sAtk = Label(start, text = "Special Attack 50", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

sAtk.place(x=600, y=180)

uAtk = Label(start, text = "Ultimate Attack 150", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

uAtk.place(x=600, y=250)

def bChar2(self):

global selection2

selection2 = 3

photo = PhotoImage(file="bCS2.png")

photo.image = photo

photo\_label = Label(player2, image=photo, bg='#FF6D9E',bd=0)

photo\_label.place(x=260, y=30)

details = Label(player2, text = "Details", bg='#68ffa9', fg='#DA6BFF', font=('Modern 20 bold'))

details.place(x=600, y=45)

nAtk = Label(player2, text = "Normal Attack 25", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

nAtk.place(x=600, y=110)

sAtk = Label(player2, text = "Special Attack 50", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

sAtk.place(x=600, y=180)

uAtk = Label(player2, text = "Ultimate Attack 150", bg='#68ffa9', fg='#DA6BFF', font=('Modern 17'))

uAtk.place(x=600, y=250)

def battle(self):

if selection2 == 0:

messagebox.showwarning("Alert", "Must choose a character") #Make sure player cannot progress if a character hasn't been chosen

else:

self.canvas = Canvas(player2, bg='#6D97FF', width=863, height=600)

self.canvas.pack()

rectangle = Canvas(self.canvas, width=863, height=402, bg='#FF6D9E', bd=0)

rectangle.place(x=0, y=439)

global check #Public attribute to be used in other methods

check = 1 #Stores 1 to keep track which screen is currently displayed on screen

top1 = Label(self.canvas, text = "Player 1", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top1.place(x=40, y=35)

self.start\_label = Label(self.canvas, text="Start!", font=('Modern 70 bold'), bg='#FF6D9E', fg='white')

self.start\_label.place(x=350, y=200)

self.start\_label.after(2000, self.start\_label.place\_forget) #Remove label after 2 seconds

self.round\_no = 1 #Store current round

self.round\_label = Label(self.canvas, text=f"Round {self.round\_no}", font=('Modern 20 bold'), bg='#FF6D9E', fg='white') #Display current round on screen

self.round\_label.place(x=405, y=140)

self.health1 = 1000

self.energy1 = 500

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Display current health on screen for player 1

lHealth.place(x=200, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Display current energy on screen for player 1

lEnergy.place(x=200, y=95)

self.health2 = 1000

self.energy2 = 500

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Display current health on screen for player 2

lHealth.place(x=540, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Display current energy on screen for player 2

lEnergy.place(x=550, y=95)

self.paused = False #Make sure game isn't paused when the battle starts

if selection1 == 1: #Check which character has been chosen

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=40, y=95)

self.orange = Orange(self.canvas, 200, 380, "oFight.png") #Display character image on screen

self.master.bind("<w>", self.orange.jump) #Bind the keys to the character class methods

self.master.bind("<a>", self.orange.move\_left)

self.master.bind("<d>", self.orange.move\_right)

self.master.bind("<KeyPress-s>", self.orange.guard)

self.master.bind("<KeyRelease-s>", self.orange.unguard)

self.master.bind('<q>',

lambda \_: Orange.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Orange.normalAtk(self))

self.master.bind('<r>',

lambda \_: Orange.specialAtk(self))

self.master.bind('<t>',

lambda \_: Orange.ultAtk(self))

elif selection1 == 2:

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=40, y=95)

self.white = White(self.canvas, 200, 380, "wFight.png")

self.master.bind("<w>", self.white.jump)

self.master.bind("<a>", self.white.move\_left)

self.master.bind("<d>", self.white.move\_right)

self.master.bind("<KeyPress-s>", self.white.guard)

self.master.bind("<KeyRelease-s>", self.white.unguard)

self.master.bind('<q>',

lambda \_: White.calcEnergy(self))

self.master.bind('<e>',

lambda \_: White.normalAtk(self))

self.master.bind('<r>',

lambda \_: White.specialAtk(self))

self.master.bind('<t>',

lambda \_: White.ultAtk(self))

elif selection1 == 3:

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=40, y=95)

self.black = Black(self.canvas, 200, 380, "bFight.png")

self.master.bind("<w>", self.black.jump)

self.master.bind("<a>", self.black.move\_left)

self.master.bind("<d>", self.black.move\_right)

self.master.bind("<KeyPress-s>", self.black.guard)

self.master.bind("<KeyRelease-s>", self.black.unguard)

self.master.bind('<q>',

lambda \_: Black.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Black.normalAtk(self))

self.master.bind('<r>',

lambda \_: Black.specialAtk(self))

self.master.bind('<t>',

lambda \_: Black.ultAtk(self))

if selection2 == 1:

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=712, y=95)

self.orange2 = Orange2(self.canvas, 675, 380, "oFight2.png")

self.master.bind("<Up>", self.orange2.jump)

self.master.bind("<Left>", self.orange2.move\_left)

self.master.bind("<Right>", self.orange2.move\_right)

self.master.bind("<KeyPress-Down>", self.orange2.guard)

self.master.bind("<KeyRelease-Down>", self.orange2.unguard)

self.master.bind('<l>',

lambda \_: Orange2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Orange2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Orange2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Orange2.ultAtk(self))

elif selection2 == 2:

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=729, y=95)

self.white2 = White2(self.canvas, 675, 380, "wFight2.png")

self.master.bind("<Up>", self.white2.jump)

self.master.bind("<Left>", self.white2.move\_left)

self.master.bind("<Right>", self.white2.move\_right)

self.master.bind("<KeyPress-Down>", self.white2.guard)

self.master.bind("<KeyRelease-Down>", self.white2.unguard)

self.master.bind('<l>',

lambda \_: White2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: White2.normalAtk(self))

self.master.bind('<n>',

lambda \_: White2.specialAtk(self))

self.master.bind('<m>',

lambda \_: White2.ultAtk(self))

elif selection2 == 3:

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=729, y=95)

self.black2 = Black2(self.canvas, 675, 380, "bFight2.png")

self.master.bind("<Up>", self.black2.jump)

self.master.bind("<Left>", self.black2.move\_left)

self.master.bind("<Right>", self.black2.move\_right)

self.master.bind("<KeyPress-Down>", self.black2.guard)

self.master.bind("<KeyRelease-Down>", self.black2.unguard)

self.master.bind('<l>',

lambda \_: Black2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Black2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Black2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Black2.ultAtk(self))

if timeLimit != 'infinite': #Check player's time limit selection

self.time\_limit = timeLimit #Set time limit to player's choice or default

self.start\_time = None

self.remaining\_time = self.time\_limit

self.start\_timer()

self.timer\_label = Label(self.canvas, text="", font=('Modern 60 bold'), bg='#FF6D9E', fg='white') #Display time on screen

self.timer\_label.place(x=410, y=40)

self.update\_timer()

else:

self.timer\_label = Label(self.canvas, text="∞", font=('Modern 60 bold'), bg='#FF6D9E', fg='white') #Otherwise do not start a timer and display infinity symbol on screen

self.timer\_label.place(x=410, y=40)

self.pause = Button(self.canvas, text='Pause', bg='#6D97FF', fg='white', bd=1, font=('Modern 15 bold'), #Display pause button screen

command = self.pause\_game, height=2, width=6)

self.pause.place(x=410,y=490)

def start\_timer(self):

self.start\_time = time.time()

def update\_timer(self):

if not self.paused: #Check if game isn't paused

self.timer\_label.config(text=str(self.remaining\_time)) #Update timer on screen

self.remaining\_time -= 1

if self.remaining\_time < 0: #Check if the time has run out

self.remaining\_time = 0 #Make sure the time doesn't go below 0 on screen

if self.health1 > self.health2: #Check which player has the most health

self.winners.append("Player 1") #Add player with the most health into the winners list

elif self.health2 > self.health1:

self.winners.append("Player 2")

else:

self.winners.append("Draw") #Otherwise add 'Draw' to the list

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white')

lose.place(x=320, y=180)

self.start\_new\_round()

self.canvas.after(1000, self.update\_timer) #Update the timer after every seconds

def pause\_game(self):

if not self.paused: #Check if the game isn't already paused

self.paused = True #Pause the game

self.show\_pause\_frame()

self.pause\_characters()

self.pause\_timer()

def resume\_game(self):

if self.paused: #Check if the game is paused

self.paused = False #Unpause the game

self.hide\_pause\_frame()

self.resume\_characters()

self.resume\_timer()

def show\_pause\_frame(self):

self.pause\_frame = Frame(self.canvas, bg="#DA6BFF", bd=100)

self.pause\_frame.place(relx=0.5, rely=0.5, anchor="center")

pause = Label(self.pause\_frame, text="Pause", bg='#6D97FF', fg='white', font=('Modern 25 bold'))

pause.pack(pady=20)

resume\_button = Button(self.pause\_frame, text="Resume", fg='white', bg='#FF746C', bd=1, font=('Modern 17 bold'),

command=self.resume\_game)

resume\_button.pack(pady=20)

retry\_button = Button(self.pause\_frame, text="Retry", fg='white', bg='#FF746C', bd=1, font=('Modern 17 bold'),

command=self.retry)

retry\_button.pack(pady=20)

inst\_button = Button(self.pause\_frame, text="How to play", fg='white', bg='#FF746C', bd=1, font=('Modern 17 bold'),

command=self.instructions)

inst\_button.pack(pady=20)

chara\_button = Button(self.pause\_frame, text="Back to character selection", fg='white', bg='#FF746C', bd=1, font=('Modern 17 bold'),

command=self.battle\_chara)

chara\_button.pack(pady=20)

start\_button = Button(self.pause\_frame, text="Back to starting page", fg='white', bg='#FF746C', bd=1, font=('Modern 17 bold'),

command=self.battle\_back)

start\_button.pack(pady=20)

def instructions(self):

howToPlay = Frame(self.canvas, bg="#6D97FF", bd=100)

howToPlay.place(relx=0.5, rely=0.5, anchor="center")

top = Label(howToPlay, text = "How to play", bg='#FF6D9E', fg='white', font=('Modern 20 bold'))

top.pack(pady=10)

left = Label(howToPlay, text="Player 1", bg='#FF6D9E', fg='white', font=('Modern 15 bold'))

left.pack(pady=10)

wasd2 = PhotoImage(file="wasd2.png")

wasd2.image = wasd2

wasd2\_label = Label(howToPlay, image=wasd2, bg="#6D97FF", bd=0)

wasd2\_label.pack(pady=10)

p1Atks = PhotoImage(file="p1Atks.png")

p1Atks.image = p1Atks

p1Atks\_label = Label(howToPlay, image=p1Atks, bg="#6D97FF", bd=0)

p1Atks\_label.pack(pady=10)

right = Label(howToPlay, text="Player 2", bg='#FF6D9E', fg='white', font=('Modern 15 bold'))

right.pack(pady=10)

arrows2 = PhotoImage(file="arrows2.png")

arrows2.image = arrows2

arrows2\_label = Label(howToPlay, image=arrows2, bg="#6D97FF", bd=0)

arrows2\_label.pack(pady=10)

p2Atks = PhotoImage(file="p2Atks.png")

p2Atks.image = p2Atks

p2Atks\_label = Label(howToPlay, image=p2Atks, bg="#6D97FF", bd=0)

p2Atks\_label.pack(pady=10)

close = Button(howToPlay, text='Close', fg='white', bg='#FF746C', bd=1, font=('Modern 20 bold'),

command = howToPlay.place\_forget, height=1, width=4)

close.pack(pady=10)

def hide\_pause\_frame(self):

self.pause\_frame.place\_forget() #Remove the pause frame from the screen

def pause\_timer(self):

self.paused = True

def resume\_timer(self):

self.paused = False

def pause\_characters(self):

self.master.unbind("<w>") #Unbind the keys of the characters so that they can't be used

self.master.unbind("<a>")

self.master.unbind("<d>")

self.master.unbind("<e>")

self.master.unbind("<r>")

self.master.unbind("<t>")

self.master.unbind("<q>")

self.master.unbind("<Left>")

self.master.unbind("<Right>")

self.master.unbind("<Up>")

self.master.unbind("<b>")

self.master.unbind("<n>")

self.master.unbind("<m>")

self.master.unbind("<l>")

self.master.unbind("<s>")

self.master.unbind("<Down>")

def resume\_characters(self):

if selection1 == 1:

self.master.bind("<w>", self.orange.jump) #Bind the keys to the characters so that they can be used again

self.master.bind("<a>", self.orange.move\_left)

self.master.bind("<d>", self.orange.move\_right)

self.master.bind("<KeyPress-s>", self.orange.guard)

self.master.bind("<KeyRelease-s>", self.orange.unguard)

self.master.bind('<q>',

lambda \_: Orange.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Orange.normalAtk(self))

self.master.bind('<r>',

lambda \_: Orange.specialAtk(self))

self.master.bind('<t>',

lambda \_: Orange.ultAtk(self))

elif selection1 == 2:

self.master.bind("<w>", self.white.jump)

self.master.bind("<a>", self.white.move\_left)

self.master.bind("<d>", self.white.move\_right)

self.master.bind("<KeyPress-s>", self.white.guard)

self.master.bind("<KeyRelease-s>", self.white.unguard)

self.master.bind('<q>',

lambda \_: White.calcEnergy(self))

self.master.bind('<e>',

lambda \_: White.normalAtk(self))

self.master.bind('<r>',

lambda \_: White.specialAtk(self))

self.master.bind('<t>',

lambda \_: White.ultAtk(self))

elif selection1 == 3:

self.master.bind("<w>", self.black.jump)

self.master.bind("<a>", self.black.move\_left)

self.master.bind("<d>", self.black.move\_right)

self.master.bind("<KeyPress-s>", self.black.guard)

self.master.bind("<KeyRelease-s>", self.black.unguard)

self.master.bind('<q>',

lambda \_: Black.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Black.normalAtk(self))

self.master.bind('<r>',

lambda \_: Black.specialAtk(self))

self.master.bind('<t>',

lambda \_: Black.ultAtk(self))

if selection2 == 1:

self.master.bind("<Up>", self.orange2.jump)

self.master.bind("<Left>", self.orange2.move\_left)

self.master.bind("<Right>", self.orange2.move\_right)

self.master.bind("<KeyPress-Down>", self.orange2.guard)

self.master.bind("<KeyRelease-Down>", self.orange2.unguard)

self.master.bind('<l>',

lambda \_: Orange2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Orange2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Orange2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Orange2.ultAtk(self))

elif selection2 == 2:

self.master.bind("<Up>", self.white2.jump)

self.master.bind("<Left>", self.white2.move\_left)

self.master.bind("<Right>", self.white2.move\_right)

self.master.bind("<KeyPress-Down>", self.white2.guard)

self.master.bind("<KeyRelease-Down>", self.white2.unguard)

self.master.bind('<l>',

lambda \_: White2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: White2.normalAtk(self))

self.master.bind('<n>',

lambda \_: White2.specialAtk(self))

self.master.bind('<m>',

lambda \_: White2.ultAtk(self))

elif selection2 == 3:

self.master.bind("<Up>", self.black2.jump)

self.master.bind("<Left>", self.black2.move\_left)

self.master.bind("<Right>", self.black2.move\_right)

self.master.bind("<KeyPress-Down>", self.black2.guard)

self.master.bind("<KeyRelease-Down>", self.black2.unguard)

self.master.bind('<l>',

lambda \_: Black2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Black2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Black2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Black2.ultAtk(self))

def retry(self):

self.resume\_game() #Resume game after it was paused

lose = Label(self.canvas, text=' ', font=('Modern 12'), bg='#6D97FF', fg='white')

lose.place(x=320, y=180)

self.winners = [] #Reset the winners list

self.actualWinner = None #Reset the overall winner

lose = Label(self.canvas, text=self.winners, font=('Modern 12'), bg='#FF6D9E', fg='white') #Reset winners on screen

lose.place(x=320, y=180)

self.health1 = 1000 #Reset the attributes

self.energy1 = 500

self.health2 = 1000

self.energy2 = 500

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold')) #Display values in attributes on screen

lHealth.place(x=200, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

self.start\_label = Label(self.canvas, text="Start!", font=('Modern 70 bold'), bg='#FF6D9E', fg='white')

self.start\_label.place(x=350, y=200)

self.start\_label.after(2000, self.start\_label.place\_forget)

self.round\_no = 1 #Reset the round number

self.round\_label = Label(self.canvas, text=f"Round {self.round\_no}", font=('Modern 20 bold'), bg='#FF6D9E', fg='white') #Display the round number on screen

self.round\_label.place(x=405, y=140)

if timeLimit != 'infinite':

if self.start\_time is not None:

self.canvas.after\_cancel(self.canvas.after(1000, self.update\_timer)) #Stop the current timer if it's running

self.start\_time = None

self.remaining\_time = self.time\_limit #Reset the timer

self.start\_timer() #Start the timer again

if selection1 == 1: #Check which character has been chosen

self.orange.canvas.delete(self.orange.imageID) #Remove the current character object from the screen

self.orange = Orange(self.canvas, 200, 380, "oFight.png") #Display character image on screen at the reset position

self.master.bind("<w>", self.orange.jump) #Bind the keys to the character class methods

self.master.bind("<a>", self.orange.move\_left)

self.master.bind("<d>", self.orange.move\_right)

self.master.bind("<KeyPress-s>", self.orange.guard)

self.master.bind("<KeyRelease-s>", self.orange.unguard)

self.master.bind('<q>',

lambda \_: Orange.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Orange.normalAtk(self))

self.master.bind('<r>',

lambda \_: Orange.specialAtk(self))

self.master.bind('<t>',

lambda \_: Orange.ultAtk(self))

elif selection1 == 2:

self.white.canvas.delete(self.white.imageID)

self.white = White(self.canvas, 200, 380, "wFight.png")

self.master.bind("<w>", self.white.jump)

self.master.bind("<a>", self.white.move\_left)

self.master.bind("<d>", self.white.move\_right)

self.master.bind("<KeyPress-s>", self.white.guard)

self.master.bind("<KeyRelease-s>", self.white.unguard)

self.master.bind('<q>',

lambda \_: White.calcEnergy(self))

self.master.bind('<e>',

lambda \_: White.normalAtk(self))

self.master.bind('<r>',

lambda \_: White.specialAtk(self))

self.master.bind('<t>',

lambda \_: White.ultAtk(self))

elif selection1 == 3:

self.black.canvas.delete(self.black.imageID)

self.black = Black(self.canvas, 200, 380, "bFight.png")

self.master.bind("<w>", self.black.jump)

self.master.bind("<a>", self.black.move\_left)

self.master.bind("<d>", self.black.move\_right)

self.master.bind("<KeyPress-s>", self.black.guard)

self.master.bind("<KeyRelease-s>", self.black.unguard)

self.master.bind('<q>',

lambda \_: Black.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Black.normalAtk(self))

self.master.bind('<r>',

lambda \_: Black.specialAtk(self))

self.master.bind('<t>',

lambda \_: Black.ultAtk(self))

if selection2 == 1:

self.orange2.canvas.delete(self.orange2.imageID)

self.orange2 = Orange2(self.canvas, 675, 380, "oFight2.png")

self.master.bind("<Up>", self.orange2.jump)

self.master.bind("<Left>", self.orange2.move\_left)

self.master.bind("<Right>", self.orange2.move\_right)

self.master.bind("<KeyPress-Down>", self.orange2.guard)

self.master.bind("<KeyRelease-Down>", self.orange2.unguard)

self.master.bind('<l>',

lambda \_: Orange2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Orange2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Orange2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Orange2.ultAtk(self))

elif selection2 == 2:

self.white2.canvas.delete(self.white2.imageID)

self.white2 = White2(self.canvas, 675, 380, "wFight2.png")

self.master.bind("<Up>", self.white2.jump)

self.master.bind("<Left>", self.white2.move\_left)

self.master.bind("<Right>", self.white2.move\_right)

self.master.bind("<KeyPress-Down>", self.white2.guard)

self.master.bind("<KeyRelease-Down>", self.white2.unguard)

self.master.bind('<l>',

lambda \_: White2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: White2.normalAtk(self))

self.master.bind('<n>',

lambda \_: White2.specialAtk(self))

self.master.bind('<m>',

lambda \_: White2.ultAtk(self))

elif selection2 == 3:

self.black2.canvas.delete(self.black2.imageID)

self.black2 = Black2(self.canvas, 675, 380, "bFight2.png")

self.master.bind("<Up>", self.black2.jump)

self.master.bind("<Left>", self.black2.move\_left)

self.master.bind("<Right>", self.black2.move\_right)

self.master.bind("<KeyPress-Down>", self.black2.guard)

self.master.bind("<KeyRelease-Down>", self.black2.unguard)

self.master.bind('<l>',

lambda \_: Black2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Black2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Black2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Black2.ultAtk(self))

def start\_new\_round(self):

if self.round\_no == rounds: #Check if the round number has reached the amount chosen by the player

self.end\_battle()

else:

self.health1 = 1000 #Reset the attributes

self.energy1 = 500

self.health2 = 1000

self.energy2 = 500

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=200, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

self.round\_no += 1 #Increment the round number by one for the next round

self.round\_label.config(text=f"Round {self.round\_no}") #Display the new round number on screen

self.start\_label = Label(self.canvas, text="", font=('Modern 70 bold'), bg='#FF6D9E', fg='white')

self.start\_label.place(x=260, y=200)

self.start\_label.config(text=f"Round {self.round\_no} !")

self.start\_label.after(2000, self.start\_label.place\_forget)

if timeLimit != 'infinite':

if self.start\_time is not None:

self.canvas.after\_cancel(self.canvas.after(1000, self.update\_timer)) #Stop the current timer if it's running

self.start\_time = None

self.remaining\_time = self.time\_limit #Reset the timer

self.start\_timer() #Start the timer again

if selection1 == 1: #Check which character has been chosen

self.orange.canvas.delete(self.orange.imageID) #Remove the current character object from the screen

self.orange = Orange(self.canvas, 200, 380, "oFight.png") #Display character image on screen at the reset position

self.master.bind("<w>", self.orange.jump) #Bind the keys to the character class methods

self.master.bind("<a>", self.orange.move\_left)

self.master.bind("<d>", self.orange.move\_right)

self.master.bind("<KeyPress-s>", self.orange.guard)

self.master.bind("<KeyRelease-s>", self.orange.unguard)

self.master.bind('<q>',

lambda \_: Orange.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Orange.normalAtk(self))

self.master.bind('<r>',

lambda \_: Orange.specialAtk(self))

self.master.bind('<t>',

lambda \_: Orange.ultAtk(self))

elif selection1 == 2:

self.white.canvas.delete(self.white.imageID)

self.white = White(self.canvas, 200, 380, "wFight.png")

self.master.bind("<w>", self.white.jump)

self.master.bind("<a>", self.white.move\_left)

self.master.bind("<d>", self.white.move\_right)

self.master.bind("<KeyPress-s>", self.white.guard)

self.master.bind("<KeyRelease-s>", self.white.unguard)

self.master.bind('<q>',

lambda \_: White.calcEnergy(self))

self.master.bind('<e>',

lambda \_: White.normalAtk(self))

self.master.bind('<r>',

lambda \_: White.specialAtk(self))

self.master.bind('<t>',

lambda \_: White.ultAtk(self))

elif selection1 == 3:

self.black.canvas.delete(self.black.imageID)

self.black = Black(self.canvas, 200, 380, "bFight.png")

self.master.bind("<w>", self.black.jump)

self.master.bind("<a>", self.black.move\_left)

self.master.bind("<d>", self.black.move\_right)

self.master.bind("<KeyPress-s>", self.black.guard)

self.master.bind("<KeyRelease-s>", self.black.unguard)

self.master.bind('<q>',

lambda \_: Black.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Black.normalAtk(self))

self.master.bind('<r>',

lambda \_: Black.specialAtk(self))

self.master.bind('<t>',

lambda \_: Black.ultAtk(self))

if selection2 == 1:

self.orange2.canvas.delete(self.orange2.imageID)

self.orange2 = Orange2(self.canvas, 675, 380, "oFight2.png")

self.master.bind("<Up>", self.orange2.jump)

self.master.bind("<Left>", self.orange2.move\_left)

self.master.bind("<Right>", self.orange2.move\_right)

self.master.bind("<KeyPress-Down>", self.orange2.guard)

self.master.bind("<KeyRelease-Down>", self.orange2.unguard)

self.master.bind('<l>',

lambda \_: Orange2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Orange2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Orange2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Orange2.ultAtk(self))

elif selection2 == 2:

self.white2.canvas.delete(self.white2.imageID)

self.white2 = White2(self.canvas, 675, 380, "wFight2.png")

self.master.bind("<Up>", self.white2.jump)

self.master.bind("<Left>", self.white2.move\_left)

self.master.bind("<Right>", self.white2.move\_right)

self.master.bind("<KeyPress-Down>", self.white2.guard)

self.master.bind("<KeyRelease-Down>", self.white2.unguard)

self.master.bind('<l>',

lambda \_: White2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: White2.normalAtk(self))

self.master.bind('<n>',

lambda \_: White2.specialAtk(self))

self.master.bind('<m>',

lambda \_: White2.ultAtk(self))

elif selection2 == 3:

self.black2.canvas.delete(self.black2.imageID)

self.black2 = Black2(self.canvas, 675, 380, "bFight2.png")

self.master.bind("<Up>", self.black2.jump)

self.master.bind("<Left>", self.black2.move\_left)

self.master.bind("<Right>", self.black2.move\_right)

self.master.bind("<KeyPress-Down>", self.black2.guard)

self.master.bind("<KeyRelease-Down>", self.black2.unguard)

self.master.bind('<l>',

lambda \_: Black2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Black2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Black2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Black2.ultAtk(self))

def end\_battle(self):

self.pause\_timer()

self.pause\_characters()

self.end = Frame(self.canvas, bg='#FF746C')

self.end.pack(fill="both", expand=True)

if check == 1: #Check if the end battle screen is opening from the battle canvas

self.end\_screen = Frame(player2, bg='#FF746C')

self.end\_screen.pack(fill="both", expand=True)

elif check == 2: #Check if the battle screen is opening from a battle that has been retried

self.end\_screen = Frame(self.end\_screen, bg='#FF746C')

self.end\_screen.pack(fill="both", expand=True)

player1\_wins = self.winners.count("Player 1") #Store how many times Player 1 has won stored in the winners list

player2\_wins = self.winners.count("Player 2") #Store how many times Player 2 has won stored in the winners list

if player1\_wins > player2\_wins:

self.actualWinner = "Player 1"

winner\_label = Label(self.end\_screen, text="Player 1 wins!", font=('Modern 50 bold'), bg='#DA6BFF', fg='white') #Declare Player 1 as the overall winner is they have more wins than Player 2

winner\_label.place(x=20, y=20)

elif player1\_wins < player2\_wins:

self.actualWinner = "Player 2"

winner\_label = Label(self.end\_screen, text="Player 2 wins!", font=('Modern 50 bold'), bg='#DA6BFF', fg='white') #Declare Player 2 as the overall winner is they have more wins than Player 1

winner\_label.place(x=20, y=20)

else:

self.actualWinner = "Draw"

winner\_label = Label(self.end\_screen, text="Draw!", font=('Modern 50 bold'), bg='#DA6BFF', fg='white') #Declare a draw otherwise

winner\_label.place(x=20, y=20)

if self.actualWinner == "Player 1":

if selection1 == 1:

winner = PhotoImage(file="oWin.png") #Display winning image of Orange Cat if Player 1 won and is Orange Cat

winner.image = winner

winner\_label = Label(self.end\_screen, image=winner, bg='#FF746C',bd=0)

winner\_label.place(x=10, y=130)

if selection2 == 1:

loser = PhotoImage(file="oLose.png") #Display losing image of Orange Cat if Player 2 lost and is Orange Cat

elif selection2 == 2:

loser = PhotoImage(file="wLose.png")

elif selection2 == 3:

loser = PhotoImage(file="bLose.png")

loser.image = loser

loser\_label = Label(self.end\_screen, image=loser, bg='#FF746C',bd=0)

loser\_label.place(x=300, y=200)

elif selection1 == 2:

winner = PhotoImage(file="wWin.png")

winner.image = winner

winner\_label = Label(self.end\_screen, image=winner, bg='#FF746C',bd=0)

winner\_label.place(x=10, y=130)

if selection2 == 1:

loser = PhotoImage(file="oLose.png")

elif selection2 == 2:

loser = PhotoImage(file="wLose.png")

elif selection2 == 3:

loser = PhotoImage(file="bLose.png")

loser.image = loser

loser\_label = Label(self.end\_screen, image=loser, bg='#FF746C',bd=0)

loser\_label.place(x=300, y=200)

elif selection1 == 3:

winner = PhotoImage(file="bWin.png")

winner.image = winner

winner\_label = Label(self.end\_screen, image=winner, bg='#FF746C',bd=0)

winner\_label.place(x=10, y=130)

if selection2 == 1:

loser = PhotoImage(file="oLose.png")

elif selection2 == 2:

loser = PhotoImage(file="wLose.png")

elif selection2 == 3:

loser = PhotoImage(file="bLose.png")

loser.image = loser

loser\_label = Label(self.end\_screen, image=loser, bg='#FF746C',bd=0)

loser\_label.place(x=300, y=200)

elif self.actualWinner == "Player 2":

if selection2 == 1:

winner = PhotoImage(file="oWin.png")

winner.image = winner

winner\_label = Label(self.end\_screen, image=winner, bg='#FF746C',bd=0) #Display winning image of Orange Cat if Player 2 won and is Orange Cat

winner\_label.place(x=10, y=130)

if selection1 == 1:

loser = PhotoImage(file="oLose.png")

elif selection1 == 2:

loser = PhotoImage(file="wLose.png")

elif selection1 == 3:

loser = PhotoImage(file="bLose.png")

loser.image = loser

loser\_label = Label(self.end\_screen, image=loser, bg='#FF746C',bd=0)

loser\_label.place(x=300, y=200)

elif selection2 == 2:

winner = PhotoImage(file="wWin.png")

winner.image = winner

winner\_label = Label(self.end\_screen, image=winner, bg='#FF746C',bd=0)

winner\_label.place(x=10, y=130)

if selection1 == 1:

loser = PhotoImage(file="oLose.png")

elif selection1 == 2:

loser = PhotoImage(file="wLose.png")

elif selection1 == 3:

loser = PhotoImage(file="bLose.png")

loser.image = loser

loser\_label = Label(self.end\_screen, image=loser, bg='#FF746C',bd=0)

loser\_label.place(x=300, y=200)

elif selection2 == 3:

winner = PhotoImage(file="bWin.png")

winner.image = winner

winner\_label = Label(self.end\_screen, image=winner, bg='#FF746C',bd=0)

winner\_label.place(x=10, y=130)

if selection1 == 1:

loser = PhotoImage(file="oLose.png")

elif selection1 == 2:

loser = PhotoImage(file="wLose.png")

elif selection1 == 3:

loser = PhotoImage(file="bLose.png")

loser.image = loser

loser\_label = Label(self.end\_screen, image=loser, bg='#FF746C',bd=0)

loser\_label.place(x=300, y=200)

else: #Otherwise if a draw display losing images for both players

if selection1 == 1:

draw1 = PhotoImage(file="oLose.png")

elif selection1 == 2:

draw1 = PhotoImage(file="wLose.png")

elif selection1 == 3:

draw1 = PhotoImage(file="bLose.png")

draw1.image = draw1

draw1\_label = Label(self.end\_screen, image=draw1, bg='#FF746C',bd=0)

draw1\_label.place(x=40, y=200)

if selection2 == 1:

draw2 = PhotoImage(file="oLose.png")

elif selection2 == 2:

draw2 = PhotoImage(file="wLose.png")

elif selection2 == 3:

draw2 = PhotoImage(file="bLose.png")

draw2.image = draw2

draw2\_label = Label(self.end\_screen, image=draw2, bg='#FF746C',bd=0)

draw2\_label.place(x=270, y=200)

self.winners = [] #Reset winners list

self.actualWinner = None #Reset the overall winner

select = Canvas(self.end\_screen, width=300, height=350, bg='#6D97FF', bd=0)

select.place(x=510, y=120)

retry\_button = Button(select, text="Retry", fg='white', bg='#DA6BFF', bd=1, font=('Modern 17 bold'),

command=self.end\_retry)

retry\_button.place(x=130, y=50)

chara\_button = Button(select, text="Back to character selection", fg='white', bg='#DA6BFF', bd=1, font=('Modern 17 bold'),

command=self.end\_chara)

chara\_button.place(x=30, y=145)

start\_button = Button(select, text="Back to starting page", fg='white', bg='#DA6BFF', bd=1, font=('Modern 17 bold'),

command=self.end\_back )

start\_button.place(x=60, y=240)

def end\_retry(self):

self.canvas = Canvas(self.end\_screen, bg='#6D97FF', width=863, height=600)

self.canvas.pack()

rectangle = Canvas(self.canvas, width=863, height=402, bg='#FF6D9E', bd=0)

rectangle.place(x=0, y=439)

global check

check = 2 #Store 2 to keep track which screen is currently displayed on screen

self.winners = [] #Reset winners list

self.actualWinner = None #Reset the overall winner

top1 = Label(self.canvas, text = "Player 1", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top1.place(x=40, y=35)

self.start\_label = Label(self.canvas, text="Start!", font=('Modern 70 bold'), bg='#FF6D9E', fg='white')

self.start\_label.place(x=350, y=200)

self.start\_label.after(2000, self.start\_label.place\_forget)

self.round\_no = 1 #Reset round number

self.round\_label = Label(self.canvas, text=f"Round {self.round\_no}", font=('Modern 20 bold'), bg='#FF6D9E', fg='white')

self.round\_label.place(x=405, y=140)

self.health1 = 1000 #Reset character attributes

self.energy1 = 500

lHealth = Label(self.canvas, text = ('Health:', self.health1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=200, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy1, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=200, y=95)

self.health2 = 1000

self.energy2 = 500

lHealth = Label(self.canvas, text = ('Health:', self.health2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lHealth.place(x=540, y=42)

lEnergy = Label(self.canvas, text = ('Energy:', self.energy2, ' '), bg='#6D97FF', fg='white', font=('Modern 20 bold'))

lEnergy.place(x=550, y=95)

self.paused = False #Make sure game isn't paused

if selection1 == 1:

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=40, y=95)

self.orange = Orange(self.canvas, 200, 380, "oFight.png") #Display character images on screen

self.master.bind("<w>", self.orange.jump) #Bind keys to character's class method

self.master.bind("<a>", self.orange.move\_left)

self.master.bind("<d>", self.orange.move\_right)

self.master.bind("<KeyPress-s>", self.orange.guard)

self.master.bind("<KeyRelease-s>", self.orange.unguard)

self.master.bind('<q>',

lambda \_: Orange.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Orange.normalAtk(self))

self.master.bind('<r>',

lambda \_: Orange.specialAtk(self))

self.master.bind('<t>',

lambda \_: Orange.ultAtk(self))

elif selection1 == 2:

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=40, y=95)

self.white = White(self.canvas, 200, 380, "wFight.png")

self.master.bind("<w>", self.white.jump)

self.master.bind("<a>", self.white.move\_left)

self.master.bind("<d>", self.white.move\_right)

self.master.bind("<KeyPress-s>", self.white.guard)

self.master.bind("<KeyRelease-s>", self.white.unguard)

self.master.bind('<q>',

lambda \_: White.calcEnergy(self))

self.master.bind('<e>',

lambda \_: White.normalAtk(self))

self.master.bind('<r>',

lambda \_: White.specialAtk(self))

self.master.bind('<t>',

lambda \_: White.ultAtk(self))

elif selection1 == 3:

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=40, y=95)

self.black = Black(self.canvas, 200, 380, "bFight.png")

self.master.bind("<w>", self.black.jump)

self.master.bind("<a>", self.black.move\_left)

self.master.bind("<d>", self.black.move\_right)

self.master.bind("<KeyPress-s>", self.black.guard)

self.master.bind("<KeyRelease-s>", self.black.unguard)

self.master.bind('<q>',

lambda \_: Black.calcEnergy(self))

self.master.bind('<e>',

lambda \_: Black.normalAtk(self))

self.master.bind('<r>',

lambda \_: Black.specialAtk(self))

self.master.bind('<t>',

lambda \_: Black.ultAtk(self))

if selection2 == 1:

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pOrange = Label(self.canvas, text = "Orange Cat", bg='#FF746C', fg='white', font=('Modern 20 bold'))

pOrange.place(x=712, y=95)

self.orange2 = Orange2(self.canvas, 675, 380, "oFight2.png")

self.master.bind("<Up>", self.orange2.jump)

self.master.bind("<Left>", self.orange2.move\_left)

self.master.bind("<Right>", self.orange2.move\_right)

self.master.bind("<KeyPress-Down>", self.orange2.guard)

self.master.bind("<KeyRelease-Down>", self.orange2.unguard)

self.master.bind('<l>',

lambda \_: Orange2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Orange2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Orange2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Orange2.ultAtk(self))

elif selection2 == 2:

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pWhite = Label(self.canvas, text = "White Cat", bg='#F7E0FE', fg='white', font=('Modern 20 bold'))

pWhite.place(x=729, y=95)

self.white2 = White2(self.canvas, 675, 380, "wFight2.png")

self.master.bind("<Up>", self.white2.jump)

self.master.bind("<Left>", self.white2.move\_left)

self.master.bind("<Right>", self.white2.move\_right)

self.master.bind("<KeyPress-Down>", self.white2.guard)

self.master.bind("<KeyRelease-Down>", self.white2.unguard)

self.master.bind('<l>',

lambda \_: White2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: White2.normalAtk(self))

self.master.bind('<n>',

lambda \_: White2.specialAtk(self))

self.master.bind('<m>',

lambda \_: White2.ultAtk(self))

elif selection2 == 3:

top2 = Label(self.canvas, text = "Player 2", bg='#FF6D9E', fg='white', font=('Modern 30 bold'))

top2.place(x=700, y=35)

pBlack = Label(self.canvas, text = "Black Cat", bg='#0D111D', fg='white', font=('Modern 20 bold'))

pBlack.place(x=729, y=95)

self.black2 = Black2(self.canvas, 675, 380, "bFight2.png")

self.master.bind("<Up>", self.black2.jump)

self.master.bind("<Left>", self.black2.move\_left)

self.master.bind("<Right>", self.black2.move\_right)

self.master.bind("<KeyPress-Down>", self.black2.guard)

self.master.bind("<KeyRelease-Down>", self.black2.unguard)

self.master.bind('<l>',

lambda \_: Black2.calcEnergy(self))

self.master.bind('<b>',

lambda \_: Black2.normalAtk(self))

self.master.bind('<n>',

lambda \_: Black2.specialAtk(self))

self.master.bind('<m>',

lambda \_: Black2.ultAtk(self))

if timeLimit != 'infinite': #Check player's time limit choice

self.start\_time = None

self.remaining\_time = self.time\_limit #Reset timer

self.start\_timer() #Start timer again

self.timer\_label = Label(self.canvas, text="", font=('Modern 60 bold'), bg='#FF6D9E', fg='white') #Display timer on screen

self.timer\_label.place(x=410, y=40)

else:

self.timer\_label = Label(self.canvas, text="∞", font=('Modern 60 bold'), bg='#FF6D9E', fg='white') #Otherwise display infinity symbol in place of timer

self.timer\_label.place(x=410, y=40)

self.pause = Button(self.canvas, text='Pause', bg='#6D97FF', fg='white', bd=1, font=('Modern 15 bold'),

command = self.pause\_game, height=2, width=6)

self.pause.place(x=410,y=490)

def end\_chara(self):

global selection1

global selection2

selection1 = 0 #Reset selections

selection2 = 0

self.end\_screen.destroy() #Go back to character selection frame from end battle frame

self.canvas.destroy()

player2.destroy()

start.destroy()

self.charaSelect1()

def end\_back(self):

global selection1

global selection2

selection1 = 0 #Reset selections

selection2 = 0

self.end\_screen.destroy() #Go back to starting screen from end battle frame

self.canvas.destroy()

player2.destroy()

start.destroy()

def battle\_chara(self):

global selection1

global selection2

selection1 = 0 #Reset selections

selection2 = 0

self.canvas.destroy() #Go back to character selection frame from battle

player2.destroy()

start.destroy()

self.charaSelect1()

def battle\_back(self):

global selection1

global selection2

selection1 = 0 #Reset selections

selection2 = 0

self.canvas.destroy() #Go back to starting screen from battle

player2.destroy()

start.destroy()

if \_\_name\_\_ == '\_\_main\_\_':

root = Tk()

game = Game(root)

game.mainloop()