Table of Contents

**Game Description 1**

**Diagram 2**

**Custom Sprites 3**

Player Sprite 3-5

Label Sprite 5-6

Monster Sprite 6-8

Boss Sprite 8-10

Gold Sprite 10

Attack Sprite 11

Arrow Sprite 12

Damage Sprite 13

Re-spawn Sprite 13

NPC Sprite 14

Mouse Sprite 14

Background Sprite 15

**Steps to Creating Game 16-17**

Mini Maple Story Adventures!

A game that I have enjoyed playing ever since I was a child was the RPG Maple Story. Today, there may be such a vast selection of games, but the original version of Maple Story will always be on the top of my list. This game will be a mini version of the 2D game and will only be one player. It will be using the same images and similar sprites as the original but the game will be implemented differently overall. The game will be both keyboard and mouse controlled. The left and right arrow keys will move the player left and right, respectively. The left ALT key will allow the user to jump and the left CTRL key will allow the user to attack. There will be a NPC in the game to sell potions to the player and each potion will restore the player to full health points and will cost 1000 gold. The player can buy these potions by pressing the spacebar or by clicking the NPC with their mouse. The game will start off allowing the user to choose either a male or female character and following, the player will choose their weapon. They have two choices: a spear or a bow and arrow. Once the stage begins, the player will start off with approximately 5000 health points and a set of monsters will spawn along with a single boss monster. Once the boss has been killed, a new set of monsters and single boss will spawn and each set will become progressively stronger (more health points). There will be approximately five sets of monsters in total and each monster will drop gold coins. The objective of the game is to kill all of these monsters and to collect the gold coins that they will drop in random amounts and ultimately, to try and beat the game. The game will end once all five sets of the monsters have been killed or when the player’s health points reach zero.

Sprite Classes

**PLAYER CLASS**

This class will define the main player for the game

Attributes

image: This attribute will represent the player and it will change depending on its current action

* Walking
* Jumping
* Attacking
* Resting

rect: This attribute will be used for positioning purposes as well as collision detections with other sprites

Lists of images (for animation purposes)

\_\_resting\_left: list containing multiple images of a player at rest, facing left

\_\_resting\_right: list containing multiple images of a player at rest, facing right

\_\_walking\_right: list containing multiple images of the player, walking right

\_\_walking\_left: list containing multiple images of the player, walking left

\_\_attacking\_right: list containing multiple images of the player attacking while facing right

\_\_attacking\_left: list containing multiple images of the player attacking while facing left

\_\_jumping\_right: This attribute will load a single image of the player jumping and facing right

\_\_jumping\_left: This attribute will load a single image of the player jumping and facing left

\_\_dead: This attribute will be a list containing multiple images of the player as a ghost

Boolean Variables (Used for animation purposes)

\_\_resting: keeps track of whether the player is at rest. It will be initialized to True

\_\_walking: keeps track of whether the player is walking. It will be initialized to False

\_\_jumping: keeps track of whether the player is jumping. It will be initialized to False

\_\_attacking: keeps track of whether the player is attacking. It will be initialized to False

\_\_dying: keeps track of whether the player is dead. It will be initialized to False

\_\_facing\_right: This attribute will be a Boolean variable to check if the player is facing right. If not, it will be set to False. It will be initialized to True

\_\_reach\_top: The purpose of this attribute is to check and see whether the player has reached the maximum y position (used for jumping). It will be initialized to False

Other Variables

\_\_counter: This attribute will be used for animation purposes. It will be used to control the number of times the image will update per unit of time

\_\_index: This attribute will also be used for animation purposes. It will be used to index the above lists

\_\_screen: This attribute will keep track of the screen variable

\_\_dx: This attribute will set the player’s x vector

\_\_dy: This attribute will set the player’s y vector

\_\_health\_points: This attribute will contain the player’s health points

\_\_take\_damage: This attribute will contain an integer value representing the amount of damage that was dealt on the player

Methods

\_\_init\_\_(screen,weapon,gender): This initializer method will take screen, weapon and gender as parameters to know which list of sprites to load (since it will differ depending on above factors). This method will load and convert all the images of the player as well as set its rect attribute. The player will be initialized on the bottom left of the screen. It will initialize the player’s health points to five thousand and load the first frame of the player in the \_\_resting\_right list of images and assign it to the image attribute.

take\_damage():This method will choose a random number between 100 to 400 and subtract the number from \_\_health\_points. It will also return the number in the \_\_take\_damage attribute

take\_damage\_boss(): This method will choose a random number between 600 to 999 and subtract the number from \_\_health\_points. It will also return the number in \_\_take\_damage

get\_position(): This method will return the position of the player

get\_health\_points(): This method will return the number in the \_\_health\_points attribute

get\_direction(): This method will return the direction that the player is facing. True if the player is facing right, False if facing left

dead(): This method will set \_\_dying to True

jump(): This method will set \_\_jumping to True, \_\_resting to False and set the bottom rect attribute to 70 less than the screen’s height

attacking(): This method will set \_\_attacking to True, \_\_resting to False, counter to 3 and index to -1 so that updating will be smooth. The bottom rect attribute will be tweaked due to unusual dimensions of the attacking frames.

attack\_finished(): This method will return True if \_\_attack\_finish is True, False otherwise. Due to unususual dimensions that result when the player is facing left, the rect will need adjustments.

moving(integer):This method takes a single integer value as a parameter and assigns it to self.\_\_dx. It also assigns self.\_\_walking attribute to True unless the user released the left or right arrow key on the keyboard; if this is the case, self.\_\_dx will be assigned a value of 0. It updates the player’s rect attribute due to unusual formats of walking images.

recover(): This method recovers the player’s \_\_health\_points to 5000

update(): This method will be responsible for repositioning the image on the screen as well as iterating through the list of images according to different Boolean variables, ultimately, causing the animations to occur. It will also check if the player has reached the end of the screen.

**LABEL CLASS**

This class will define a label to display the amount of HP the player has, the amount of gold the player has collected, and the current stage

Attributes

image: This attribute will represent the label

rect: This attribute will be used to position the label. It will be positioned on the top of the screen

\_\_font: This attribute will create a custom font object and set its size

\_\_amount: This attribute will contain the amount of gold the player has acquired. It will be initialized to zero

\_\_health\_points: This attribute will contain an integer value that represents the player’s health points

\_\_stage: This attribute will contain an integer value that represents the stage number the player is currently on

Methods

\_\_init\_\_(screen, health\_points, value, monsters): This initializer method will take 4 parameters: health\_points to set the label for the amount of HP the player has remaining, value to know how much the gold picked up was worth, and monster\_count to know how many monsters there are remaining. It will initialize the rect attribute to the center-top of the screen

set\_health\_points(health\_points): This method will set the \_\_health\_points attribute to health\_points.

set\_amount(value): This method will add value to the \_\_amount attribute

set\_stage(stage\_number): This method will add 1 to \_\_stage

spend\_gold(): This method subtracts 1000 gold from \_\_amount. If the user does not have enough gold, do not allow action to take place. This method returns True if the user has enough, False, if not.

update(): This method will update the text that will be displayed on the top of the screen, according to the \_\_stage, \_\_amount, and \_\_health\_points attributes

**MONSTER CLASS**

This class will define a single monster for our game

Attributes

image: This attribute will be represented by an image of the monster

rect: This attribute will be used to position the sprite on the screen as well as check for collisions with the user

Lists of images (for animation purposes)

\_\_moving: This attribute will contain a list of images of the first monster

\_\_moving2: This attribute will contain a list of images of the second monster

\_\_moving3: This attribute will contain a list of images of the third monster

\_\_moving4: This attribute will contain a list of images of the fourth monster

\_\_moving5: This attribute will contain a list of images of the fifth monster

\_\_dying: This attribute will contain a list of images of the first monster passing away

\_\_dying2: This attribute will contain a list of images of the 2nd monster passing away

\_\_dying3: This attribute will contain a list of images of the 3rd monster passing away

\_\_dying4: This attribute will contain a list of images of the 4th monster passing away

\_\_dying5: This attribute will contain a list of images of the 5thmonster passing away

Other attributes

\_\_health\_points: This attribute will keep track of how many health points the monster has

\_\_dx: This attribute will represent the x vector of the monster

\_\_counter: This attribute will be used for animation purposes. It will be used to control the number of times the image will update per second

\_\_index: This attribute will also be used for animation purposes. It will be used to index the above lists

\_\_current\_monster: This attribute will contain a single integer value that will determine which monster to set the image attribute to

\_\_take\_damage: This attribute will contain an integer value representing the amount of damage that was dealt on the monster

\_\_position: This attribute will contain a position for the monster

Boolean Variables

\_\_dead: This attribute will keep track of whether the monster is dead. Initialize to False

\_\_finished: Keeps track of whether the animation of the monster dying is finished

\_\_walking: This attribute will keep track of whether the monster is walking

\_\_respawn

\_\_respawn\_finished

Methods

\_\_init\_\_(screen, integer): This initializer method will load multiple lists of monster animations. It will create a monster on a random part of the screen on the ground. It will take screen and integer as parameters and initialize \_health\_points to 2000 and \_\_dx to 1 or 0 or -1 (So not all of the monsters will move in the same direction); this will be dependent on the integer parameter. It will set the image attribute to the first image of the first monster either moving left or right or standing still (also depending on the integer parameter).

take\_damage():This method will choose a random number between 200 to 1000 and subtract the number from \_\_health\_points. The number in \_\_take\_damage will be returned

get\_position(): This method returns the position of the monster

next\_monster(): This method will assign the current list of images used for the monster to the next one. It will assign \_\_finished to False, \_\_dead to False and add one to \_\_current\_monster. It will also get the new rect of the monster and position the monster on the screen at a random location that would have already been stored in the \_\_positition attribute that was assigned in the respawn() method.

respawn(): This method will assign a random position for the monster and return this position so that the Respawn (effect) sprite will know where to position itself.

update(): This method will be responsible for repositioning the image on the screen as well as iterating through the list of images according to different Boolean variables, ultimately, causing the images to act as an animation. It will also check if the monster has died or reached the end of the screen.

**BOSS MONSTER CLASS**

This class will define a single boss monster for the game

Attributes

image: This attribute will be represented by an image of the monster

rect: This attribute will be used to position the sprite on the screen as well as check for collisions with the user

Lists of images (for animation purposes)

\_\_going\_right1: list of images of the boss moving right

\_\_going\_left1: list of images of the boss moving left

\_\_dying\_right1: list of images of the boss dying facing right

\_\_dying\_left1: list of images of the boss dying facing left

\*\*Repeat for five other bosses (\_\_going\_right2 and etc.)

Other variables

\_\_health\_points: This attribute will keep track of how many health points the monster has

\_\_dx: This attribute will represent the x vector of the monster

\_\_counter: This attribute will be used for animation purposes. It will be used to control the number of times the image will update per unit of time

\_\_index: This attribute will also be used for animation purposes. It will be used to index the above lists of images

\_\_dying\_index: This attribute will be used for indexing the list of \_\_dying images. It will be initialized to 0

\_\_take\_damage: This method will hold a number that will be the monsters taken damage. It will be initialized to 0

\_\_current\_boss: This attribute will contain a single integer value that will determine which boss to set the image attribute to. It will be initialized to 0

Boolean Variables

\_\_dead: This attribute will keep track of whether the monster is dead. Initialize to False

self.\_\_direction\_left: This attribute will keep track of which way the monster will be going. True means going left, False means going right. It will be initialized to True

\_\_finished: Keeps track of whether the animation of the monster dying is finished. It will be initialized to False

Methods

\_\_init\_\_(screen): This initializer method will create the first boss monster at the bottom right of the screen. It will take map and screen as a parameter and initialize \_\_health\_points to 50000 and \_\_dx to -3 It will set the image attribute to the first image of the boss moving left.

take\_damage(): This method will choose a random number between 1000 to 5000 and subtract the number from \_\_health\_points. This damage will be returned

moving(): This method will assign \_\_moving attribute to True

dead(): This method will return True if the animation of the monster dying has finished

next\_boss(): This method will assign the current list of images used for the boss to the following one. It will assign \_\_finished to False, \_\_dead to False and add one to \_\_current\_boss. It will also get the new rect of the monster and position the monster on the screen at a random location that would have already been stored in the \_\_positition attribute that was assigned in the respawn() method.

respawn(): This method will assign a random position for the monster and return this position so that the Respawn (effect) sprite will know where to position itself.

change\_direction(going\_left): This method will change the direction of the boss monster and will take going\_left as a parameter which will be either True or False (Used to have boss follow player)

get\_position(): This method returns the position of the monster

update(): This method will be responsible for repositioning the boss sprite on the screen as well as iterating through the list of images according to the two different Boolean variables, causing the animations to occur. It will also check if the boss has died or reached the end of the screen, and if so it will it will iterate through a list of images of it passing away will and reverse the \_\_dx attribute, respectively.

**GOLD CLASS**

This class will define gold coins

Attributes

image: This attribute will be represented by a single image of gold

rect: This attribute will position the gold and check for collisions with the player sprite

\_\_value: This attribute will be a number in the range of 50-200 and will be the value of this coin

\_\_screen: keeps track of the screen variable

Methods

\_\_init\_\_(): This initializer method will create a single coin object and load and assign its image. It will position the gold outside of the screen.

reset(position): This method updates the gold sprite. The set\_position parameter will hold either True or False. If it is True, that means that the gold and player rects collided, if this is the case, the image will be positioned off the screen. If False, it will assign a new, random value to the gold that is between 50 to 200 and position rect.center of the coin on the screen in accordance with the x vector of the position parameter which will be where the monster last died. The y vector of the coin will be on the floor of the map.

get\_valule(): This method will return the number in the \_\_value attribute

**ATTACK CLASS**

This class will represent the animation of the player’s attack.

Attributes

image: This attribute will represent the effect of attack

rect: This attribute will position the attack and check for collisions with the monster sprites. It will be positioned according to where the player is

\_\_attack\_left: This attribute will be a list of images of the attack’s effect going left

\_\_attack\_right: This attribute will be a list of images of the attack’s effect going right

\_\_counter: This attribute will be used for animation purposes. It will be used to control the number of times the image will update per unit of time

\_\_index: This attribute will also be used for animation purposes. It will be used to index the above lists

\_\_finished: The purpose of this attribute is to keep track of whether the animation of the attack’s effect has been finished. It will be initialized to False

\_\_screen: Used to keep track of the screen parameter

Methods

\_\_init\_\_(screen): This method takes screen as a parameter. It will load the images of the effect and set the image attribute to the first image inside the \_\_attack\_right list of images. It will be positioned outside of the screen where it cannot be seen.

start(): This method takes an integer and a position as parameters. The integer will be either 1 or 2, if it is 1 this means that the player is attacking and facing left. If it is 2 the player is facing right. It will assign self.\_\_position to the position parameter to position the effect where the player is

finish(): This method will return True if the animation is finished

update(): This method will update the image attribute causing it to animate. If there is an IndexError, it means that the effect finished animating. If this is the case, it will position its rect outside the screen to avoid collisions with other sprites and set \_\_finished to True.

**ARROW CLASS**

This class will show the animation of an arrow shooting (only if player is using bow)

Attributes

image: This attribute will represent the attack

rect: This attribute will position the attack and check for collisions with the monster sprites. It will be positioned according to where the player is

\_\_going\_right: This attribute will be either True or False. If the attack is going right, it will be True. If going left, it will be False

\_\_attack\_right: This attribute will be a list of images of the arrow going right

\_\_attack\_left: This attribute will be a list of images of the arrow going left

\_\_dx: This attribute will be initialized to -3 or 3 depending on the \_\_going\_right attribute

\_\_counter: This attribute will be used for animation purposes. It will be used to set the number of times the image will update

\_\_index: This attribute will also be used for animation purposes. It will be used to index the above lists

Methods

\_\_init\_\_(screen): This method takes screen as a parameter and load all the images of the arrow, going both right and left.

update\_arrow(position, integer): This method takes an integer and position as parameters. The integer will be either 1 or 2. This integer will also be used to set the \_\_dx attribute of the arrow. If it is 1 this means that the player is attacking and facing left, therefore \_\_dx will be assigned a value of -3. If it is 2 the player is facing right and \_\_dx will be assigned a value of 3. It will assign self.\_\_position to the position parameter to position the effect where the player for the update method

hit(): This method will be called when the arrow and a monster collides. It will position the arrow outside the screen where it will no longer interfere with other sprites and set the \_\_dx attribute to 0

update(): This method will update the image attribute causing it to animate. It will also move the image across the screen by adding the number stored in the \_\_dx attribute to the rect of the sprite

**DAMAGE CLASS**

This class will display damage

Attributes

image: This attribute will represent the damage dealt on player or monster

rect: This attribute will position the image

\_\_font: This attribute will create a font object in a custom font

Methods

\_\_init\_\_(): This class will create a custom font object

update\_damage(position, damage, attack\_who): This method will take the position of the player or monster as a parameter and set the rect.midbottom of the image to the top of the player as well as render the font to the number in the damage variable. The attack\_who parameter will determine the colour of the text.

**RESPAWN CLASS**

This class will define an effect that will be shown when a monster or boss is respawn into the game

Attributes

image: This attribute will be represented by the effect

rect: This attribute will position the effect of the sprite

\_\_list: This attribute will contain a list of frames of the respawning effect

Methods

\_\_init\_\_(): This initializer method will load the respawning effect and assign its image to the first image of the list. It will position the effect outside of the screen.

reset(position): This method updates the respawn sprite. It will place the image where the monster is respawning.

update(): This method will update the frame of the effect (for animations)

**NPC CLASS**

This class will create a NPC for the game

Attributes

Image: will represent the NPC

rect: used to position the sprite

\_\_npc\_images: holds a list of images of the NPC

\_\_counter: This attribute will be used for animation purposes. It will be used to control the number of times the image will update per unit of time

\_\_index: This attribute will also be used for animation purposes. It will be used to index the above lists

Methods

\_\_init\_\_(): This initializer method will position the image of the NPC on the screen

update(): updates animation of NPC

**MOUSE CLASS**

This class will define a custom cursor for the game!

Attributes

image: image of the cursor

rect: will be assigned to the position of the mouse

\_\_normal: This attribute will hold a single image of the mouse

\_\_click: This attribute will hold a single image of the mouse being clicked

\_\_counter: This attribute will be used for animation purposes. It will be used to set the number of times the image will update

Methods

\_\_init\_\_(position): This method will create the mouse cursor and set its position to the current position of the mouse

click(): This method will change the mouse to the to an image of the mouse clicking.

release(): This method will change the image attribute back to the normal image

update(): This method will reposition the cursor image to the position of the mouse.

**BACKGROUND CLASS**

This class will define 5 backgrounds for the game

Attributes

image: image of the background

rect: positions the background

list\_of\_images: attribute that contains a list of 5 loaded images

\_\_index: used to index the list

Methods

\_\_init\_\_(): This method will load the 5 different backgrounds and assign the image attribute with the first image of the five

next\_map(): This method will add one to the \_\_index attribute and load the next map

Steps to Creating Maple Story

1. Create a simple box sprite and figure out how to make it jump
2. Make the box sprite move
3. Change the image of the sprite to a player. Add animations for the player and get the frame-changing timing correct for walking
4. Get the rest of the player animations to work (jumping, attacking, & resting)
5. Create a single boss monster, get it to move correctly and check to see whether it reverses direction when it reaches the end of the screen
6. Create multiple monsters with animations and see if the game will lag
7. Try and get damage to show the way I want it show
8. Check for collision detection between player and monster, figure out how to remove the monster from the game when it runs out of HP. See what happens when you hit multiple monsters at once and see if separate damages for each monster show
9. Add the label sprite to display the current status of the game
10. Add gold coins and collision detection between player and the coins. If the two collide, add the amount of gold that it was worth to the total amount of gold the player has
11. Create an NPC Sprite. Make it so that if the player presses the space bar they will recover HP as long as they have enough gold
12. Add a custom cursor
13. Have new monsters appear after the current ones have been wiped out. Figure out how to get them to drop the same coins that were used with the previous set of monsters and recycle the sprites.
14. Spawn a new boss when the player clears the stage
15. Add a transition screen between stages.
16. Make the dragon monsters fly
17. Add a background and change it every time a boss is killed.
18. Make all the images needed for the game have a transparent background (approx. 600)
19. Add background music
20. Add sound effects
21. Add another weapon and allow let the user to choose their gender
22. Get bow and arrow to work as intended
23. Add a ‘first’ screen, with instructions and character/weapon selection
24. Try and get it to show how much was in the gold that the player picked up. Show this in the label
25. Add an underwater scene as a bonus round
26. Since damages overlap, making them difficult to discern, try and make it so that the other damage appears on top of the other