# Cardboard Minecraft Pygame project

## Analysis

### Project Background

Within a school environment, the amount of pressure on students to perform in exams is immense. With this comes a heavy volume of stress on college students on a day-to-day basis.

I believed from my own experiences and my experiences from the people around me is that without being able to release this stress the performance for students in exams would drop largely. This is because when I went without playing games for long periods of time, it resulted in my concentration during revision sessions being lower and I also had difficulty in sleeping if I do not give myself a chance to relax after heavy brain work.

Therefore, I did a bit of research and the papers that I looked at came to a similar conclusion.

Referencing this paper:

https://d1wqtxts1xzle7.cloudfront.net/54975414/6._EFFECT_OF_PERCIEVED_ACADEMIC-1-with-cover-page-v2.pdf?Expires=1649708683&Signature=F88a2OfRUM3MDi10XjtLjmGvZGwZcHQKiPUTKRD6-24Vol~vuJDJM5lh7wAjrgZCwW~bFpwK62OxOg4DGJS~051uLmdhfFeNJo~i8klr-qDcViPl~5iIL6NT3BVMwlENNEhEjIGKcqUXgBLXn1I7AuwFquS27n5OqPDHvGu5ro2p3U8O2OB4BYPVdiPuMuwe8KMXJw9818yOii-y25Wi0fKOhy5pkcVjeHrFOmaXt9d-vq2B-wAeG6ZH-ExeU8ly9J7vI6Sj8aFUSPBDWV8jXSNDXeromYzNPNY9il~3MVemyNMJ9C5Ek7lN~Is82KhOXre1COgnRcoOG1MaV51k-w__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

This paper which researched the effect of academic stress of student's performance, concluded that stress is a major factor in exam performance and so if we do not allow for students to be able to cool down some time after doing large amount of heavy brain work then there would be effects in the exam results.

Furthermore, it is widely accepting that for releasing stress and cooling down, one of the best ways to do this is by playing games.

Reference:

https://www.healthygamer.gg/blog/do-video-games-reduce-stress#:~:text=Video%20games%20give%20you%20instant,unwind%20after%20a%20long%20day.

This article discusses how video games are a great way to release stress as they provide complete immersion into an activity in such ways that watching TV and relaxing cannot. They also provide instant gratification, which is the opposite of delayed gratification, which what the real world works

upon. Although, instant gratification has some bad effects as it prioritises the current state and ignores future states, it is a necessary evil for students to be able to cope with exam stress as it provides us a refreshing break from our demanding lives.

Instant gratification also allows us to overcome setbacks in our everyday life as they set small and attainable goals which can be met more simply than large goals which allows for a smoother mental state rather than the normal falling and dipping as stressful events come and go during our student life.

The area I am investigating is game development, this is because games which can be played school are limited to the ones installed on the system which are mostly logic, problem solving and introduction to coding. However, these games do not provide a large amount of immersion and release from college stress.

Further research suggests that the best type of game to play to release stress would be a single player game as it means that there is no one to argue with or bring your spirits down and you can simply enjoy yourself my relaxing on your own. Furthermore, multiplayer games come with your teammates relying and depending on you to perform which means the situation is identical to school as you once again have stress to perform but this time in a game environment.

Also, the best type of game to release stress would be a game in which you feel free to adventure, express your creativity and be challenged in a way that you feel engaged with every moment. The game genre that fits these categories the most, is a sandbox game. This allows for the player to take control of their experience and gives many paths a player can take to progression.

Reference:

https://www.lifewire.com/joys-of-single-player-minecraft-3968220

This article backs up the points of playing single player games to release stress and that stress release is proportional to the immersion in a game or activity and so a sandbox games ranks the highest.

My project: Cardboard Minecraft, is a Terraria inspired 2D (sideview) single-player PC game. With the genre being sandbox, exploration, and Role-Playing Game.

Sandbox games such as Minecraft/Terraria are extremely fun games with involve an addicting gameplay loop of exploring new areas, mining, and harvesting, killing enemies and bosses. Then collecting rewards and upgrading the player.

## Target Audience

This game will be targeted towards college students who already enjoy gaming to relax as gaming is only relaxing if you know what you are doing, learning how to game on a pc from scratch would be a stressful experience and so there would be no point in playing the game.

Therefore, target audience age will be 16-21.

Since it is aimed at college students who already play games, the game will be challenging in nature to increase the level of immersion, which as stated earlier is proportional to the volume of stress released.

However, this will be an “casual mode” feature which allows you to simply enjoy the game stress free as there will be fewer monster summons and they will be less aggressive. Life regeneration will also be increased. -Reduces challenge on user

Furthermore, there will be “creative mode” which will allow you to access everything in the game. This will let you test all the items in the game without having to obtain all the materials and build whatever you want. -Removes the time taken to obtain the items in the game, this will allow people who don’t want to play the game seriously to play as well.

## Research

### Pygame Syntax/ initial framework

In research for this game in looked at tutorials on https://realpython.com/ and learned how to make the framework for a simple side scrolling game where the character stays in the centre and all the sprites move instead. I also referred to https://www.pygame.org/docs/ to learn the correct syntax I will need to create my game.  Additionally, I have occasionally referred to stack overflow or Wikipedia for information.

### Market research

The bulk of my research my research was by playing Terraria and understanding how the game works.

From this research I discovered that my game would have to be very simple compared to its inspiration as the fully-fledged sandbox game would never get completed in the time frame. From this I was able to categorise and refine what specifically I would add to the game.

I liked that there is only one screen in the entire gameplay and so reduces the complexities of menus which most RPGs have in the current market.

Fig 1- This one screen consists of the HUD/ inventory/ storage/ armour and accessory slots.

Screenshots from terraria game:

Yellow-Features that I may look to add to my game

Red-Features that I am not including (to simplify game)


*Figure 1*


*Figure 2*

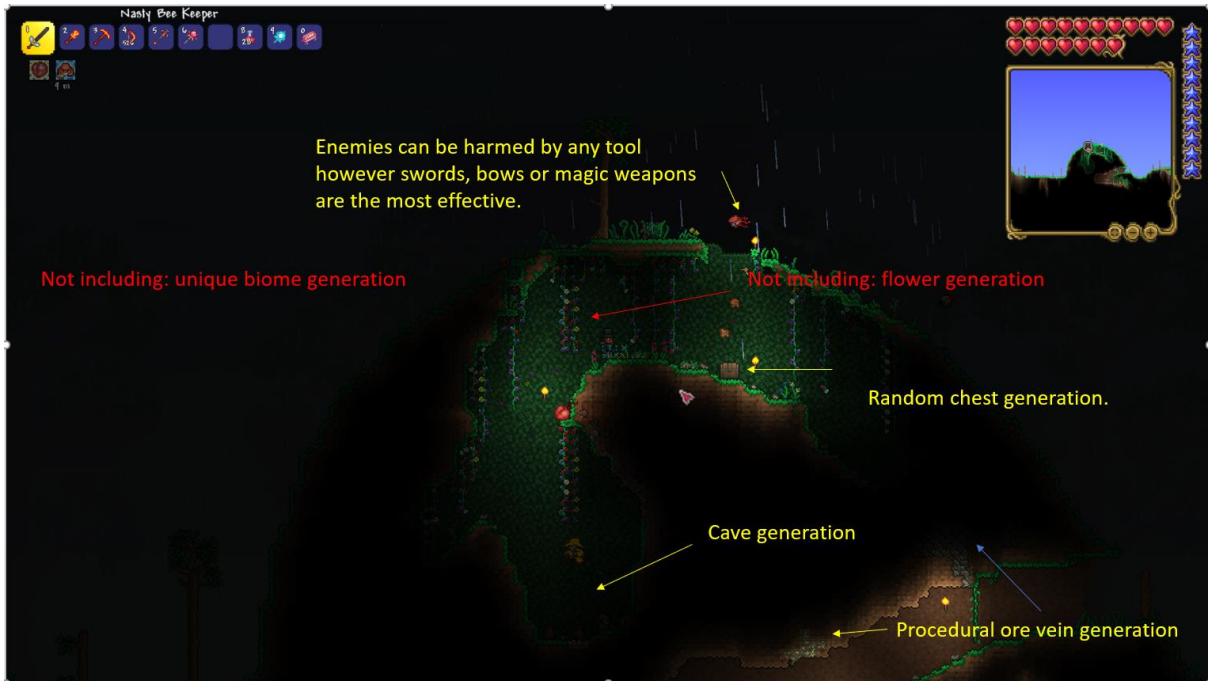



*Figure 3*


*Figure 4*

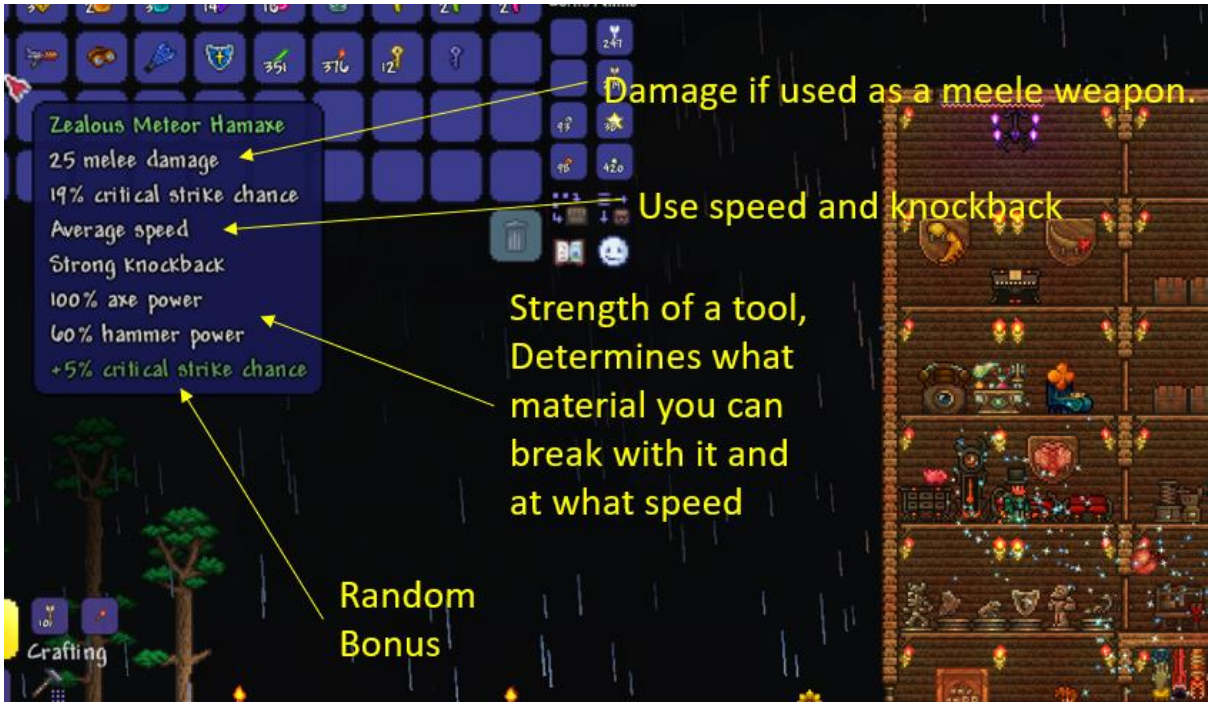



*Figure 4*


*Figure 5*

*Figure 6: arrow type projectile*

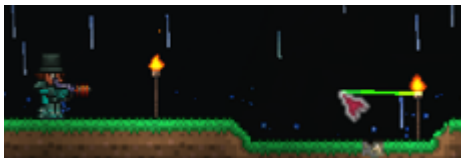


*Figure 7: laser type projectile*

*Figure 9: magic missile projectile*

*Figure 10: Summons sprites*


*Figure 11: Menu screen*


*Figure 12: World select*


*Figure 13: World Creation*

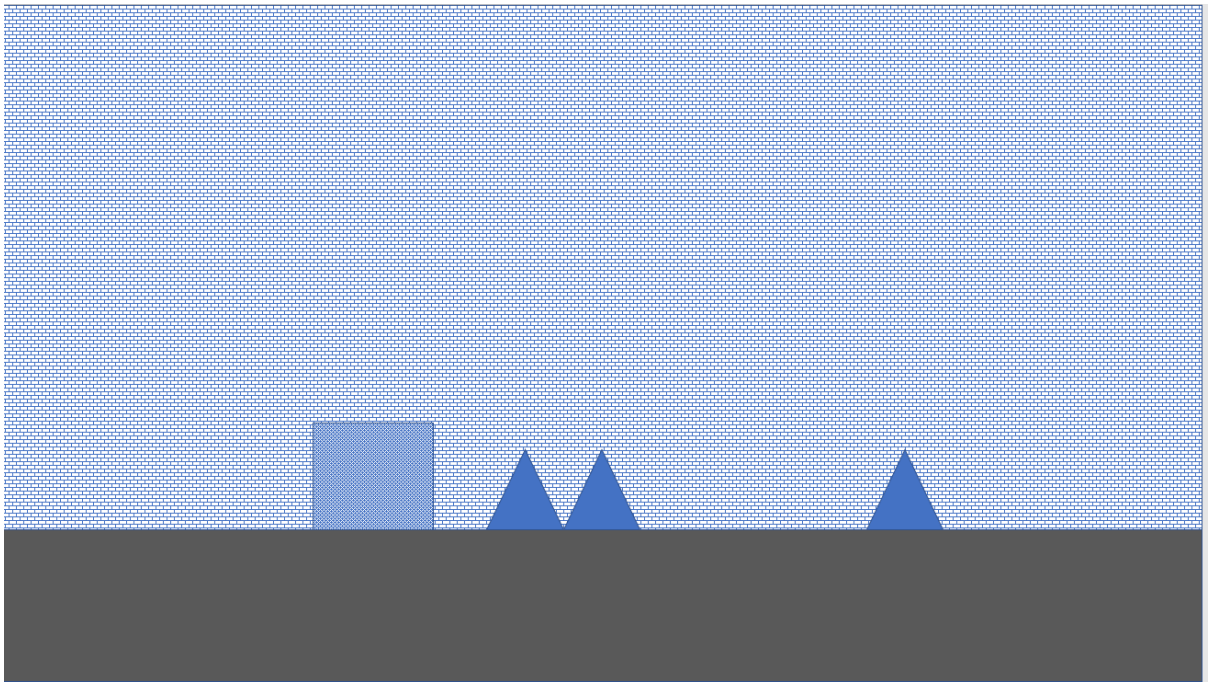

*Figure 14: Settings menu*

## Pygame Practice

### *Vertices Rush*

To first get a feel for pygame and how to use to simulate physics I created a prototype game "vertices rush" based on the game "Geometry dash."

### Modelling



Model created using PowerPoint.

I planned it out to look something like this

## Research

For the making of this game, I didn't know how to use Pygame at all and so used a program off https://realpython.com/ as a template for the framework.

```
# Simple pygame program
 2
 3# Import and initialize the pygame library
 4import pygame
 5pygame.init()
 6
 7# Set up the drawing window
 8screen = pygame.display.set_mode([500, 500])
 9
10# Run until the user asks to quit
11running = True
12while running:
13
14    # Did the user click the window close button?
15    for event in pygame.event.get():
16        if event.type == pygame.QUIT:
17            running = False
18
19    # Fill the background with white
20    screen.fill((255, 255, 255))
21
22    # Draw a solid blue circle in the center
23    pygame.draw.circle(screen, (0, 0, 255), (250, 250), 75)
24
25    # Flip the display
26    pygame.display.flip()
27
28# Done! Time to quit.
29pygame.quit()
```

For research of the events and motion of the character I also pinched this basic game off stack overflow.

https://stackoverflow.com/questions/16551009/gravity-in-pygame

```
import pygame, sys
from pygame.locals import *

pygame.init()

FPS = 30
fpsClock = pygame.time.Clock()

DISPLAYSURF = pygame.display.set_mode((400, 300), 0, 32)
pygame.display.set_caption("Jadatja")

WHITE = (255, 255, 255)
catImg = pygame.image.load("images/cat.png")
catx = 10
caty = 10
movingRight = False
movingDown = False
movingLeft = False
movingUp = False

while True: #main game loop

   #update
   for event in pygame.event.get():
   if event.type == KEYDOWN:
      if event.key == K_RIGHT:
         #catx += 5
         movingRight = True
         movingLeft = False
      elif event.key == K_DOWN:
         #caty += 5
         movingDown = True
         movingUp = False
      elif event.key == K_LEFT:
         #catx -= 5
         movingLeft = True
         movingRight = False
      elif event.key == K_UP:
         #caty -= 5
         movingUp = True
         movingDown = False

   if event.type == KEYUP:
      if event.key == K_RIGHT:
         movingRight = False
      if event.key == K_DOWN:
         movingDown = False
      if event.key == K_LEFT:
         movingLeft = False
      if event.key == K_UP:
         movingUp = False


   #actually make the player move
   if movingRight == True:
      catx += 5
   if movingDown == True:
      caty += 5
   if movingLeft == True:
```

```
        catx -= 5
    if movingUp == True:
        caty -= 5


    #exit
    for event in pygame.event.get():
        if event.type == KEYUP:
            if event.key == K_ESCAPE:
                pygame.quit()
                sys.exit()

        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    #draw
    DISPLAYSURF.fill(WHITE)
    DISPLAYSURF.blit(catImg, (catx, caty))



    pygame.display.update()
    fpsClock.tick(FPS)
```

## Objectives:

### 1) Background

-Background moving backward at a slower rate and looping back around

### 2) Floor

-Background moving backward at a faster rate and looping back around

### 3) Player

-Pressing a mouse button will cause the player (square) to jump up and rotate

-If player lands on a spike, it will die

-speed of player speed up over time (causes all other sprites to move faster instead)

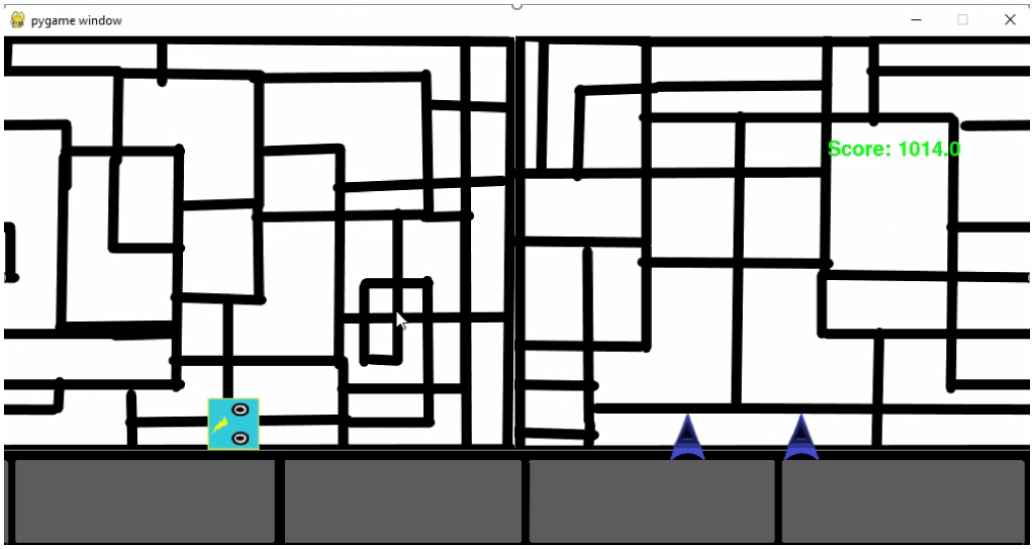
### 4) Spike

-randomly generated

-2 spikes per floor loop

-moves backward as if glued to the floor

## Evaluation




-All the objectives have been met

-However, a flaw in this game was that sometimes spikes can spawn in ways in which it is impossible to jump over. Therefore, a more intelligent procedural generation was needed to fix this problem.

-This program has set the floor to the ground y position and so doesn't collide with objects, this means that it doesn't allow for the expansion of adding higher floors

-Furthermore, the lack of rect type colliders meant that the spikes were modelled by a cross:

 Hitbox in red

-The character would have to completely recreated to add different character modes in Geometry dash such as the ship:

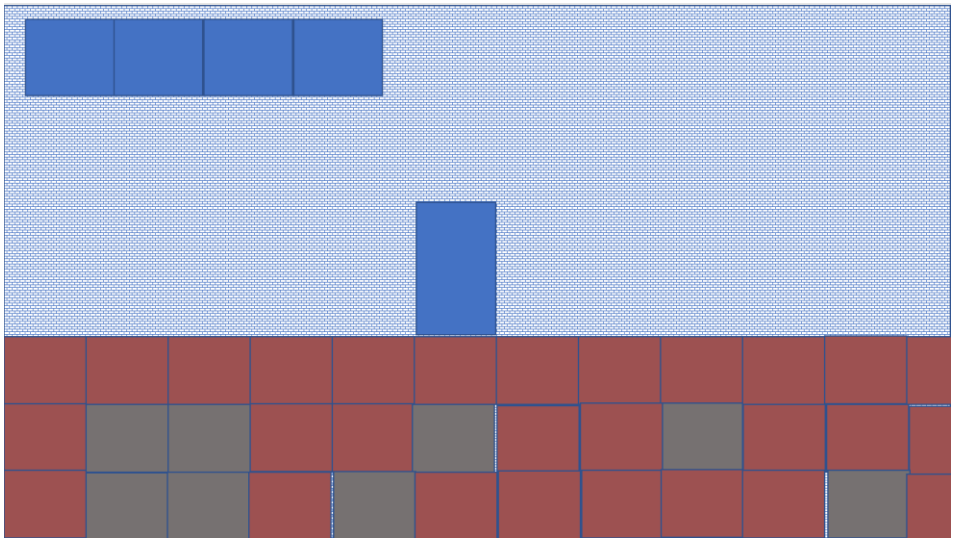
Image from StickNodes.com

-It would be difficult to meet the level of complexity I want with this level of game. This is because there is not much room for development. For example, you only click to jump and so there is no room to display my ability to create multidirectional motion

-To solve this problem, I decided to develop a more open, sandbox game such as terraria, which allowed me more freedom in development.

*Cardboard Minecraft lite*

## Modelling



## Research

To research for this prototype, I used https://realpython.com/ again. Specifically: https://realpython.com/courses/pygame-primer/ I looked at the outline and code to create a 2D side-scroller game on real python.com

## Objectives

### 1) Tiles

-The floor is randomly generated from tiles of two types

-The character can break the tiles and collect them, then they can place them again

-These rectangles will be replaced with the textures which I am borrowing from the Terraria game

### 2) Motion

-character moves left right using the “a” key and “d” key, the character jumps in the form of a parabola using the “space” key.

### 3) Camera

-The camera moves with the character and so if the character breaks the blocks below and moves down then the camera will follow

### 4) Collision

-The character collides with each tile as they are all separate entities

### 5) Inventory

-Simple inventory system: the blue tiles in the air represent the inventory hotbar which moves with the player

-Upon breaking tiles, they will go straight into the inventory

## Evaluation


All the objectives were met however there were severe problems that meant I had to abandon the game I was making and switch to a new model

-The player can move left and right and can jump (player moves down under the force of gravity)

-The player will collide with all sprites as if there are individual tiles

-The player will be able to break blocks with mouse click and collect in inventory

-The player can scroll through the inventory and place blocks with mouse click

-The tiles are randomly generated, generated rarer materials at lower y levels. Procedural generation algorithm shown on the right→

```
BIList = []
xCount = 40 * 6
yCount = 20 * 5
xOffset = ((xCount - 40) / 80) * 1000
yOffset = (((yCount) / 40) -1) * 500
for x in range(xCount):
    for y in range(yCount):
        Roll = random.randint(1, 1000)

        if y <= 11 + ((yOffset/500)*20):
            BI = Air()

        elif y <= 15 + ((yOffset/500)*20):
            if Roll < 50:
                BI = Stone()
            else:
                BI = Dirt()

        elif y < 25 + ((yOffset/500)*20):
            if Roll < 50:
                BI = TinOre()
            elif Roll < 250:
                BI = Stone()
            else:
                BI = Dirt()

        elif y <= 40 + ((yOffset/500)*20):
            if Roll <= 25:
                BI = TungstenOre()
            elif 25 < Roll <= 75:
                BI = LeadOre()
            elif 75 < Roll <= 175:
                BI = TinOre()
            elif 175 < Roll <= 425:
                BI = Stone()
            else:
                BI = Dirt()

        elif y <= 60 + ((yOffset/500)*20):
            if Roll <= 25:
                BI = PlatinumOre()
            elif 25 < Roll <= 75:
                BI = TungstenOre()
            elif 75 < Roll <= 175:
                BI = LeadOre()
            elif 175 < Roll <= 425:
                BI = TinOre()
            else:
                BI = Stone()


        elif y <= 80 + ((yOffset/500)*20):
            if Roll <= 25:
                BI = MithrilOre()
            elif 25 < Roll <= 75:
                BI = PlatinumOre()
            elif 75 < Roll <= 175:
                BI = TungstenOre()
            elif 175 < Roll <= 425:
                BI = LeadOre()
            else:
                BI = Stone()

        BI.rect.x = x * 25 - xOffset
        BI.rect.y = y * 25 - yOffset
        BI.worldX=BI.rect.x
        BI.worldY=BI.rect.y
        BIList.append(BI)
```


-However instead of making a camera which would shift the offset of all blocks I was changing the position of all blocks when the player moved, this was quite processor intensive, and it also created a chance for things to go wrong.

-My jump sequence was flawed as it didn't complete the parabola and so when it touched the ground again the y position of blocks in the block array was changed.

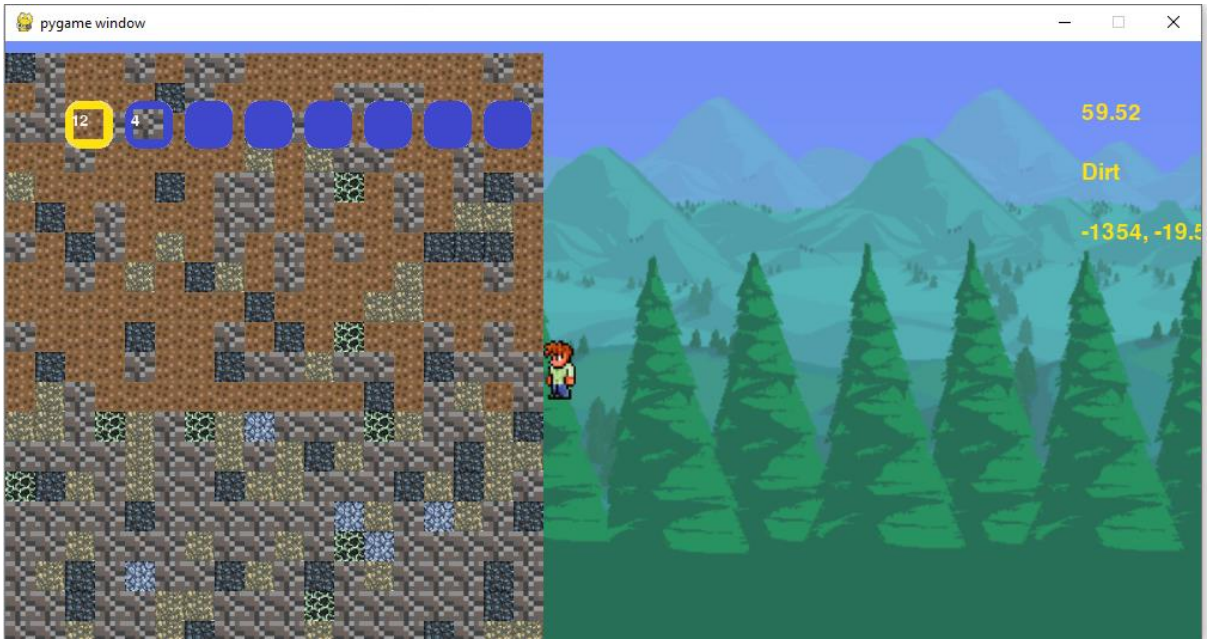
-This resulted in the blocks above being squashed into the top blocks when I went below and then came back up


-Furthermore, the collision was also flawed as it allowed me to cling onto walls if I moved into them and allowed me to catch tiles with the top of my characters rectangle

-This was because when I was moving into a wall it would set the velocity to zero and push out of the wall, however it would also push upward as the character is also intersecting with the floor in that moment, allowing you to stick onto or slide down walls

-Collision also meant that the player would sink into the floor as the velocity is made zero on contact with the floor, but the players y position didn't change after intersecting with the floor tiles.

-This resulted in the case of character being propelled off the map if jumping down with a high y velocity, as the side colliders intersect with the tiles and won't stop giving a repulsive velocity until I am off the map




## Conclusion

### *What I have decided to add from my research:*

Menu (including the loading and saving of worlds) – no fun in a game where you progress is not saved. This menu needs to only have 4 screens (for simplicity): home, world select, world create settings.

Hotbar/Inventory (including proper descriptions when you hover over it) – collecting stuff is fun and the description allows a player to learn to play the game easily.

Health– provides challenge

Crafting window – allows you make stuff from the things you collect

Setting access – allows you to customise game

Armour – you can choose to obtain armour if you need so or go without it if you have confidence in your skills and are willing to take a risk. Provides extra choice for the player

Darkness – game wouldn't be fun if you could see every ore on your screen, fun comes with exploration and discovery.

Placeable tiles such as work benches effecting crafting-adds depth and immersion

Procedural generation of ores and caves- adds replicability into the game, ores will not be in the same place every time.

Weapons-sword– killing enemies is a great way to release stress

Tools-making use of the correct tools adds a layer of difficulty

Day/Night cycle – provides depth

Projectiles – look cool

Different Difficulty levels-more options for the user

### *What I have decided to omit from my research:*

Coins and ammo- unneeded complexity

Minimap – performance issues as you must take a scan of the current map and constantly update the minimap

Vanity/accessories –too large for this project

Status effects – unneeded complexity

Doors, Tables, chairs – doesn't add anything to the game

Lighting – performance issues as it requires the game to constantly use path finding algorithms

NPC and their housing – too large for this project

Flower / biome generation – too large for this project

Potions – too large for this project

Armour effects – unneeded complexity

Random weapon/tool effects – too large for this project

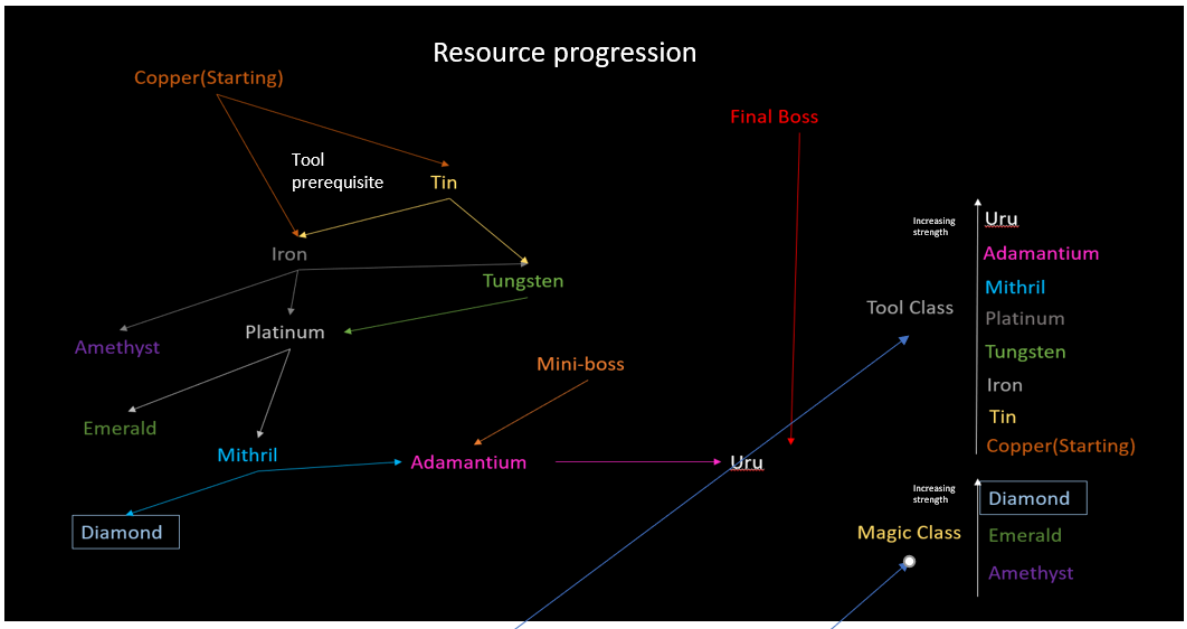
Events- too large for this project

Bow/magic/summoning-Too large for this project

Mana – too large for this project

## Initial Modelling

## Progression




This is showing the progression of tool strength.

The arrows show that you need to obtain that material.
For example: with a copper pickaxe you can mine iron and tin.
To obtain adamantium you need mithril tools and to have killed the mini-boss

All tools can mine the ores below their strength level

This is showing the progression of magic item strength

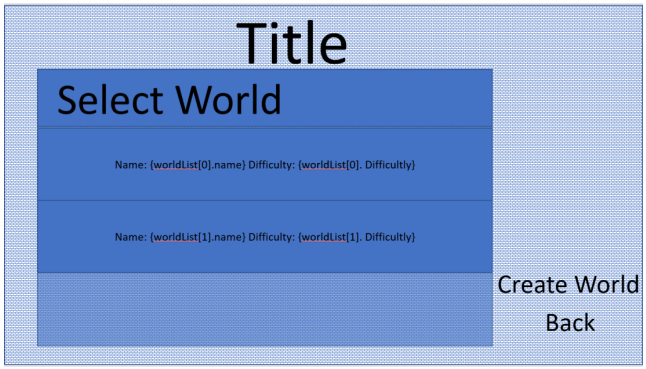
## Menu

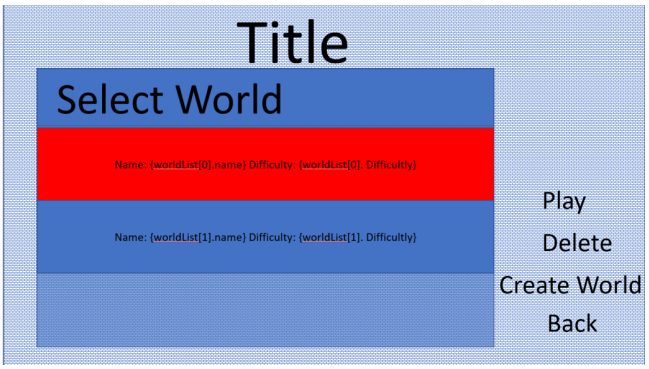
Based on fig 11-14

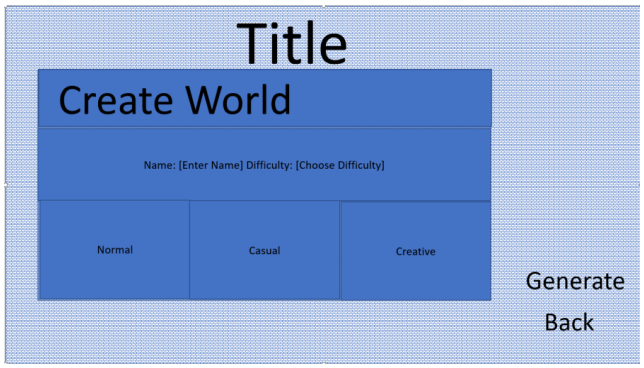


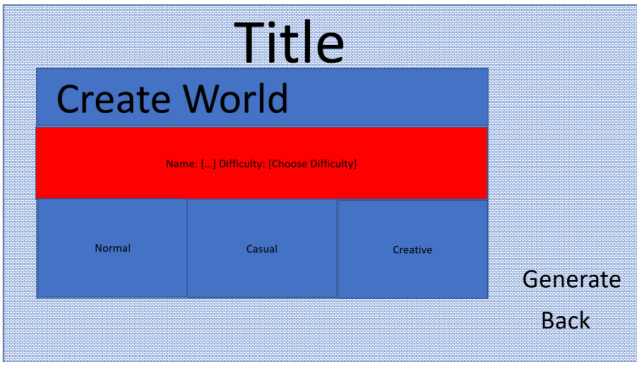
All buttons will highlight when you however over the rectangle for that button

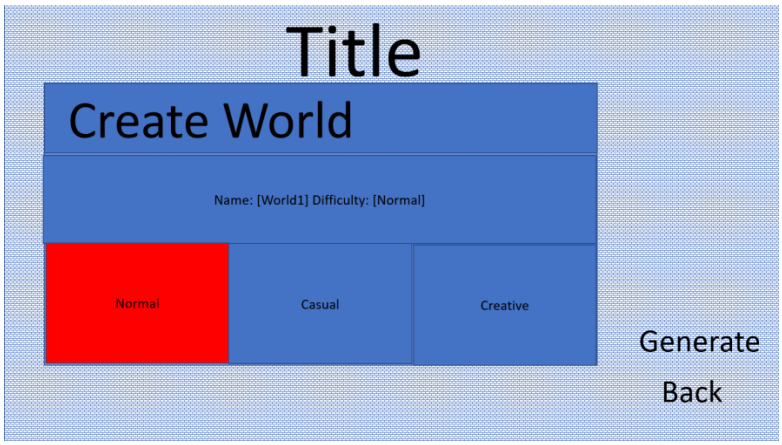

Title
Settings
Hotbar Scrolling
Left
Right
Volume
0.25
Back


Title
Select World
Name: {worldList[0].name} Difficulty: {worldList[0]. Difficultly}
Name: {worldList[1].name} Difficulty: {worldList[1]. Difficultly}
Create World
Back

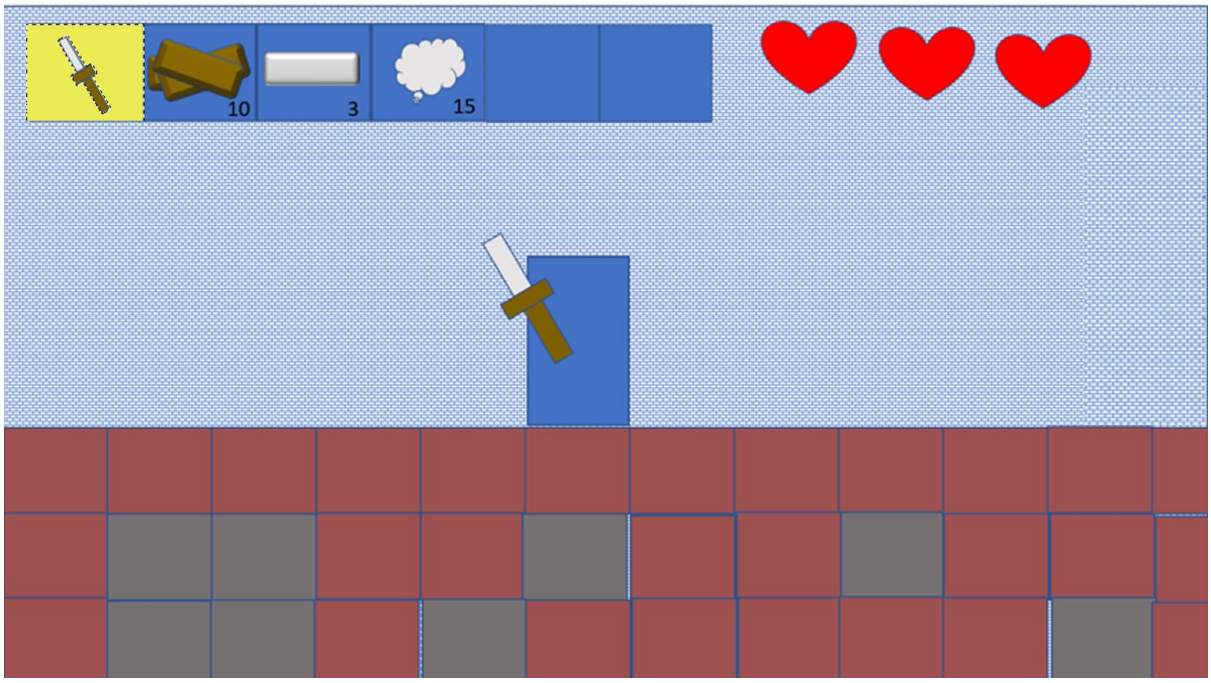

Title
Select World
Name: {worldList[0].name} Difficulty: {worldList[0]. Difficultly}
Name: {worldList[1].name} Difficulty: {worldList[1]. Difficultly}
Play
Delete
Create World
Back


Title
Create World
Name: [Enter Name] Difficulty: [Choose Difficulty]
Normal
Casual
Creative
Generate
Back

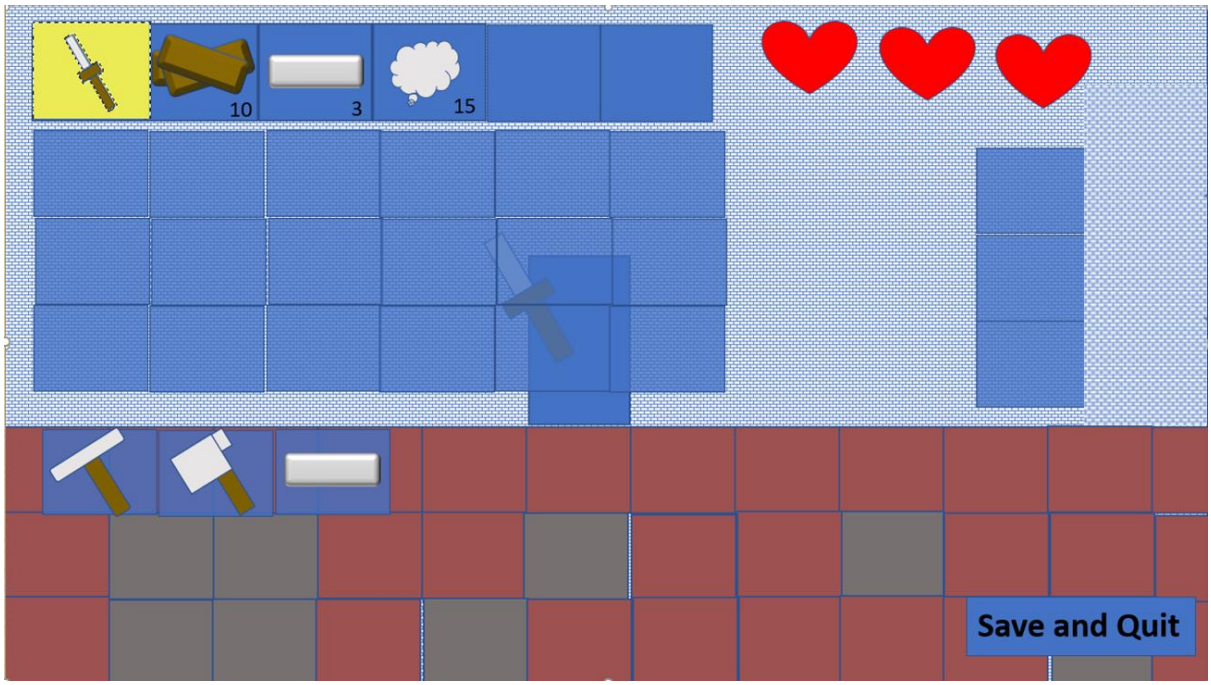

Title
Create World
Name: [...] Difficulty: [Choose Difficulty]
Normal
Casual
Creative
Generate
Back


Title
Create World
Name: [World1] Difficulty: [Normal]
Normal
Casual
Creative
Generate
Back

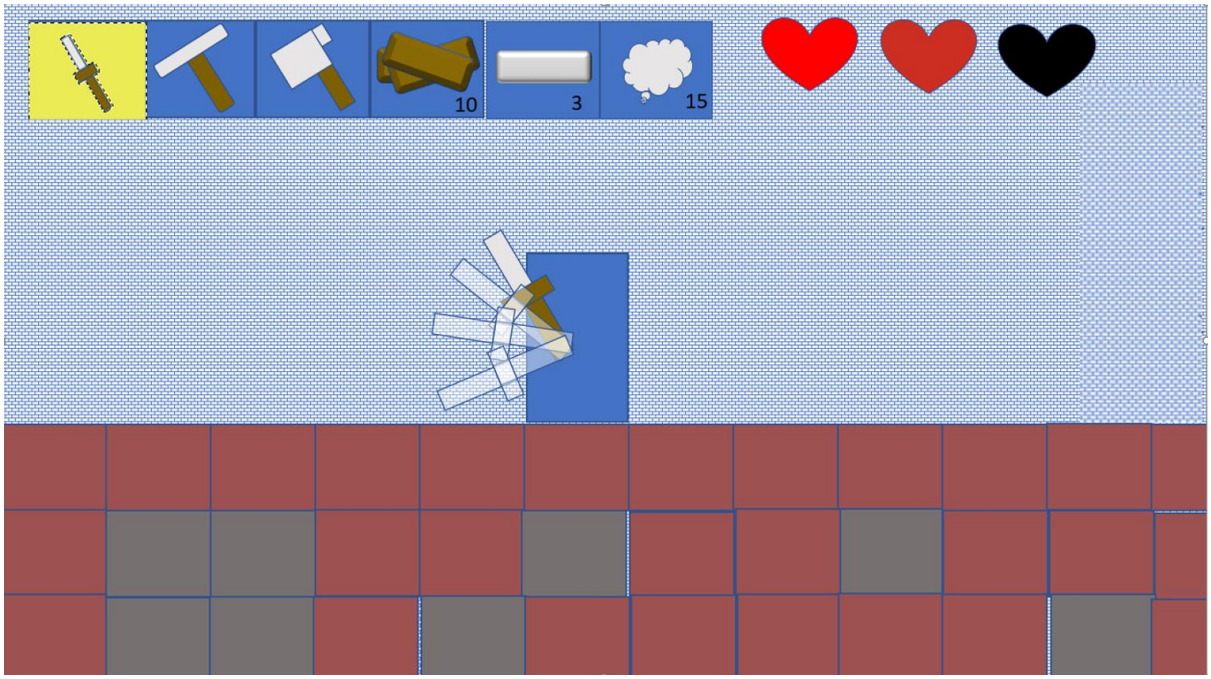
## Game Interface




*Normal HUD State*




*Full HUD opened (game paused)*




*Example swing/health bar/ animation*

## Objectives

### 1) Menus:

-View modelling - Menu

-settings, you will be able to toggle different options such as which way the inventory scrolls.
-world generation, all procedural generation takes place, and all world and player progression are saved to a world file.
-select difficulty
-loading and saving to and from save file.

### 2) Player motion and Camera:

- Player moves around in both x and y, horizontal motion is done with “A” and “D” and vertical (jumping) is with the space bar (parabolic motion.) This is because people who play PC games would be used to these controls.

-As the player moves the screen scrolls all the other active sprites to keep them in the center of the screen

### 3) Collision:

-player collides with all other sprites apart from “background” objects (objects you pass through such as trees and walls)
-on collision the player is moved to the outside of the rectangle collision which makes it look like the player is at a standstill on collision

## 4) Tiles:

-View Fig 2 and modelling- Game Interface

-placed and destroyed with the correct tools and taking some time or number of hits to destroy

-Blocks

-Different workbenches

-Walls (background object)

## 5) Materials:

- View modelling- Game Interface

-Obtained by breaking blocks and other tiles

-Used in crafting

## 6) Procedural Generation:

-View Fig 3

-Ores, generated in block veins and at different probabilities based on y position, this means that ore which is rarer will be generated more frequently the lower down you go.

-Caves

-Trees

-background will change according to the environment you are in, for example sky for above ground, dirt for underground and stone for the deep underground.

-lighting: blocks around many other blocks should be shrouded in darkness or it would be too easy to get resources

## 7) HUD

-View Fig 1 and modelling- Game Interface

-Health bar; will display your health using several hearts and fractions of a heart as a key to how much health you have left. Each heart will be 20hp. This is more immersive than simple using text.

-inventory screen: hotbar (scroll to select an item on hotbar or use a number key), storage, armour

-crafting screen; displays craftable items

## 8) Player inventory and health:

-View Fig 1 and 2 and modelling- Game Interface

-Hotbar consisting of 8 slots

-Item inventory of 40 slots

-Equipment inventory with 3 slots for armour.

-max health of 100hp

-if player is dealt damage, then health goes down, cooldown to regen after taking damage

-damage sources: falling, enemies

-regenerates over time proportional to fraction of health you have left

-regens linearly

## 9) Items:

-View Fig 4 and 5

-Weapons; sword

-Tools; pickaxe, axe, hammer

-Armour; varying strengths, give you defence (damage reduction) – I will do this if I get time as it is not a very important objective since it is only damage reduction.

-all items will do damage when swung but using weapons is the most effective

-projectiles created by items:

## 10) Crafting:

-Fig 6

-Crafting recipes appear in your crafting section of the inventory when you have the correct items to craft something.

-You can only craft certain things next to certain workbenches

## 11) Enemies:

-Different types of foes appear in the world of varying strengths, I.e.: a slime is weak but a skeleton may be strong.

-Enemies may fire bullets/spells

## 12) Progression:

-View Modelling Progression

-The game should have an innate clear progression to be designed in a way that whatever you do you make progress toward something and that if you want to make meaningful progress quickly then you have the option.

### 13) Events:
-Day/Night cycle

### 14) Animations:
-All objects, items and tools will have an animation set

### 15) Difficulty:
-normal

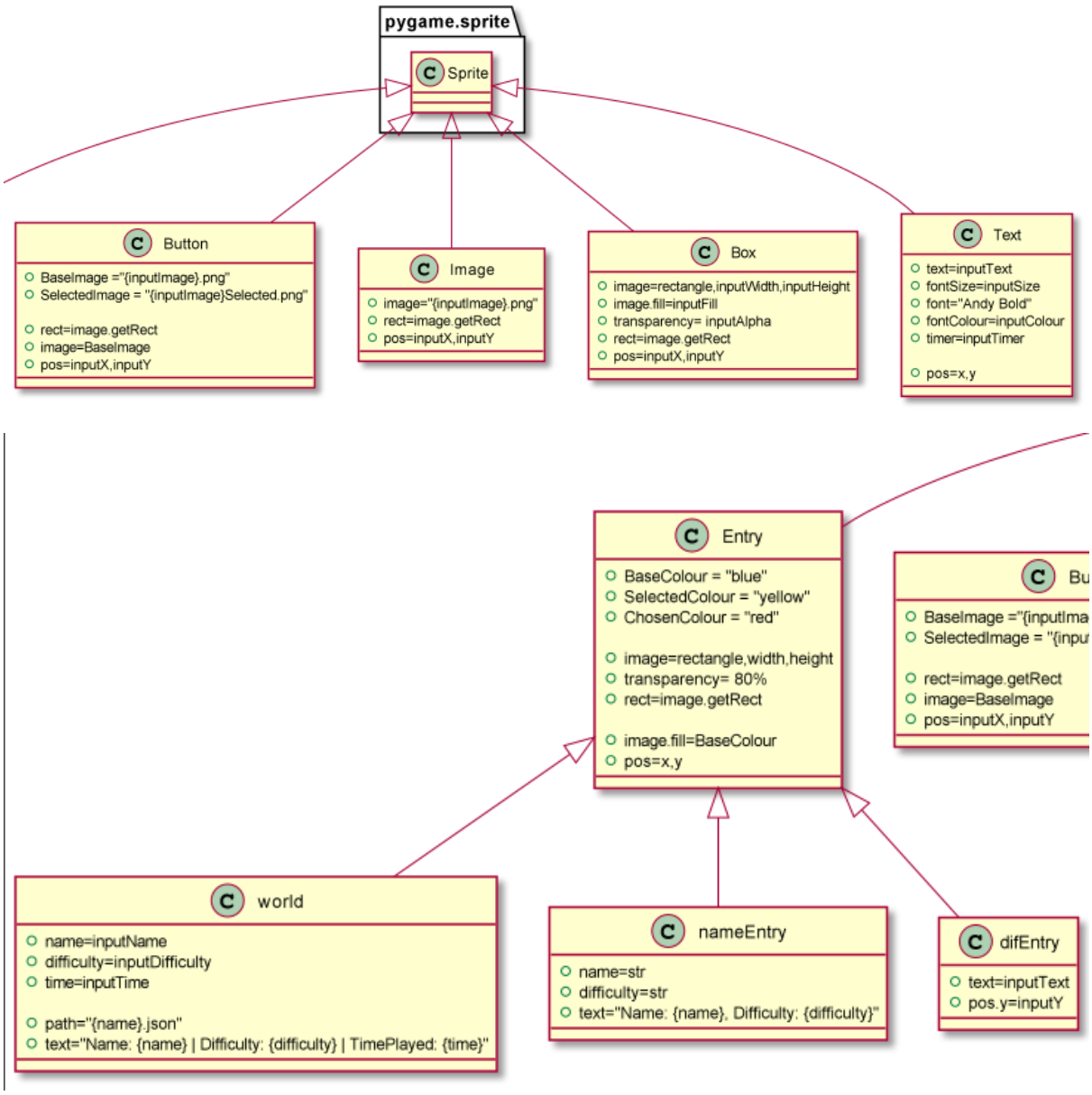
-casual: reduced monster summons, increased health regens, halves all damage dealt by enemies

-creative: can fly and have no clip, can craft anything without materials but the correct bench is still required, infinite damage reduction

### 16) Optimisation:
-Game must run at a minimum 30FPS, preferred is 60FPS

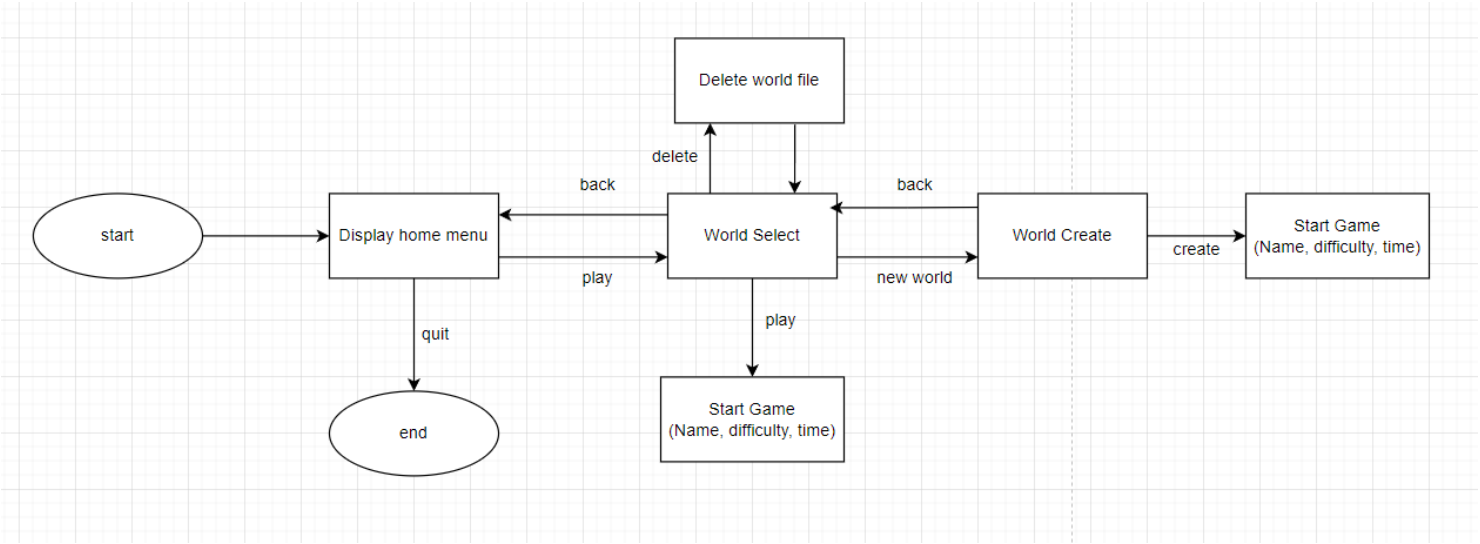
## Design

### Objective 1 Menus
Class diagram for menu



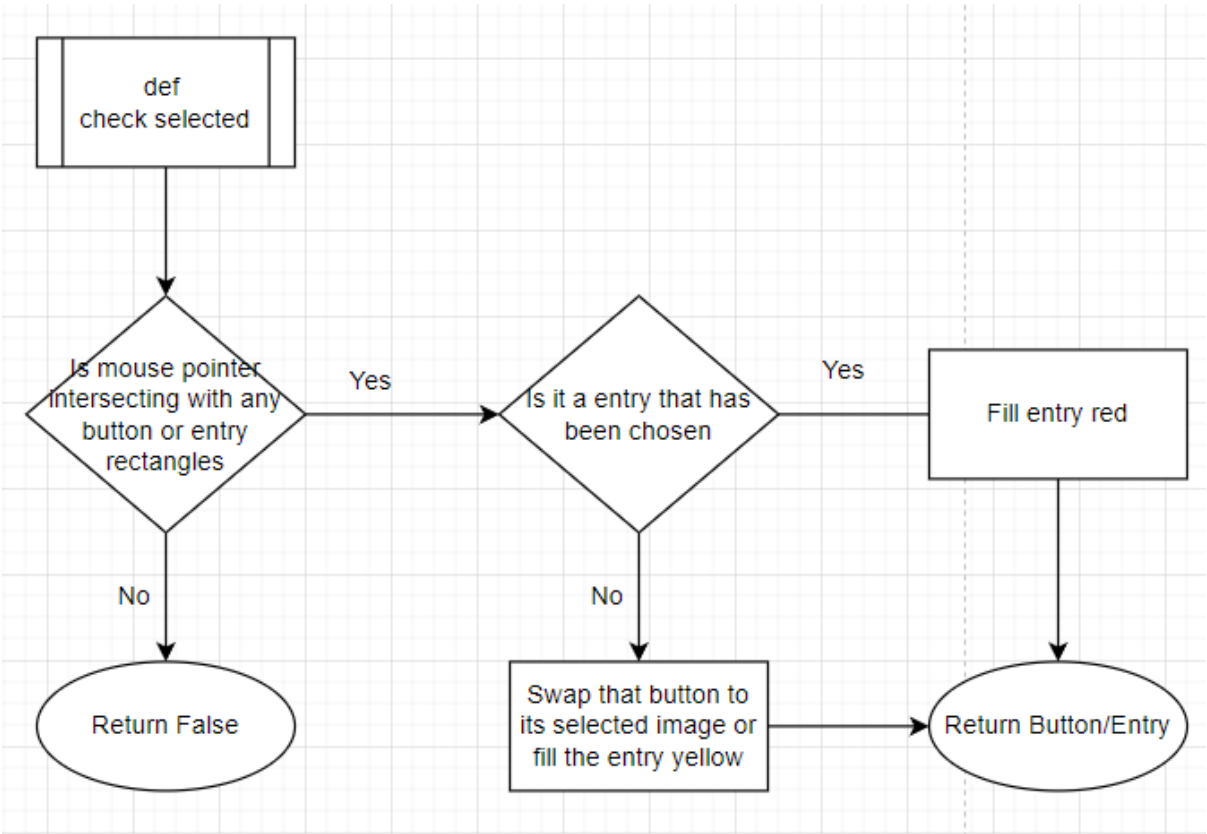

Flow chart:

Outline flow of how buttons work in the menu

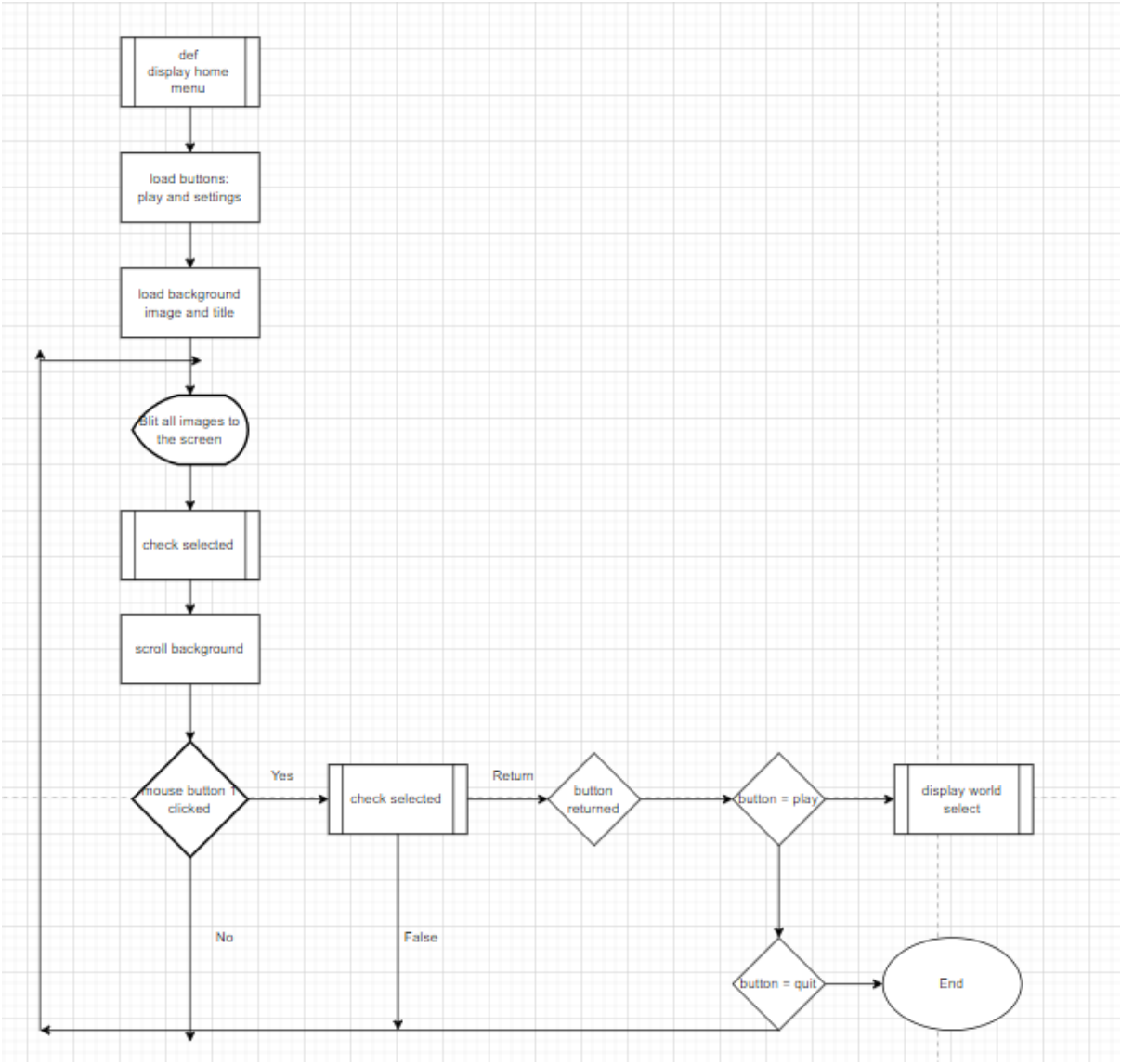



Detailed diagrams for each subroutine:

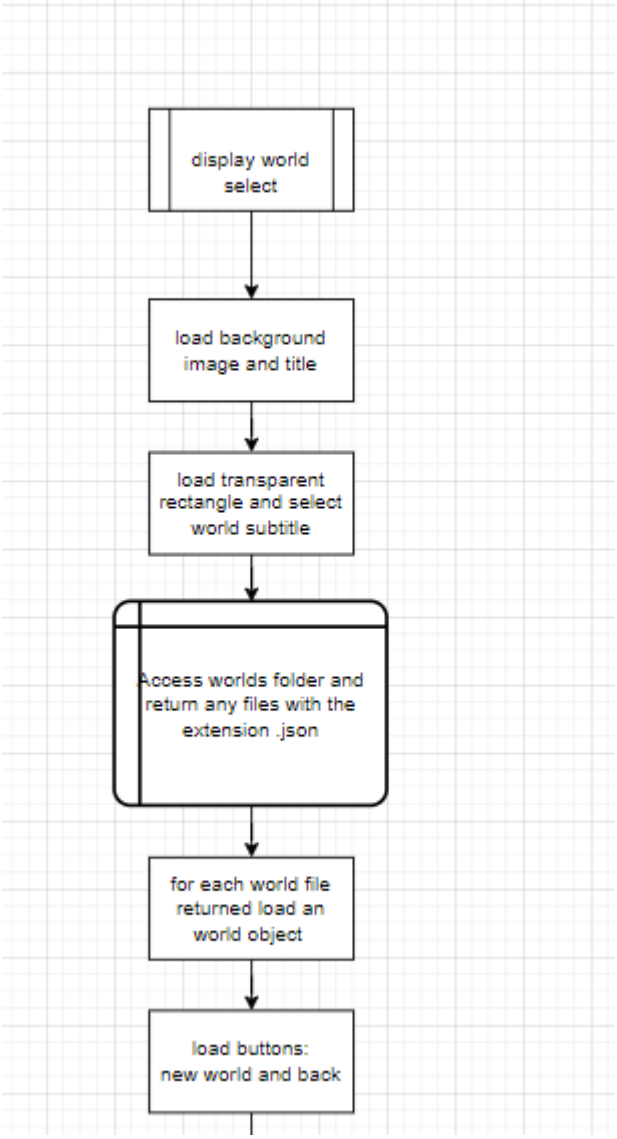
Check Selected:

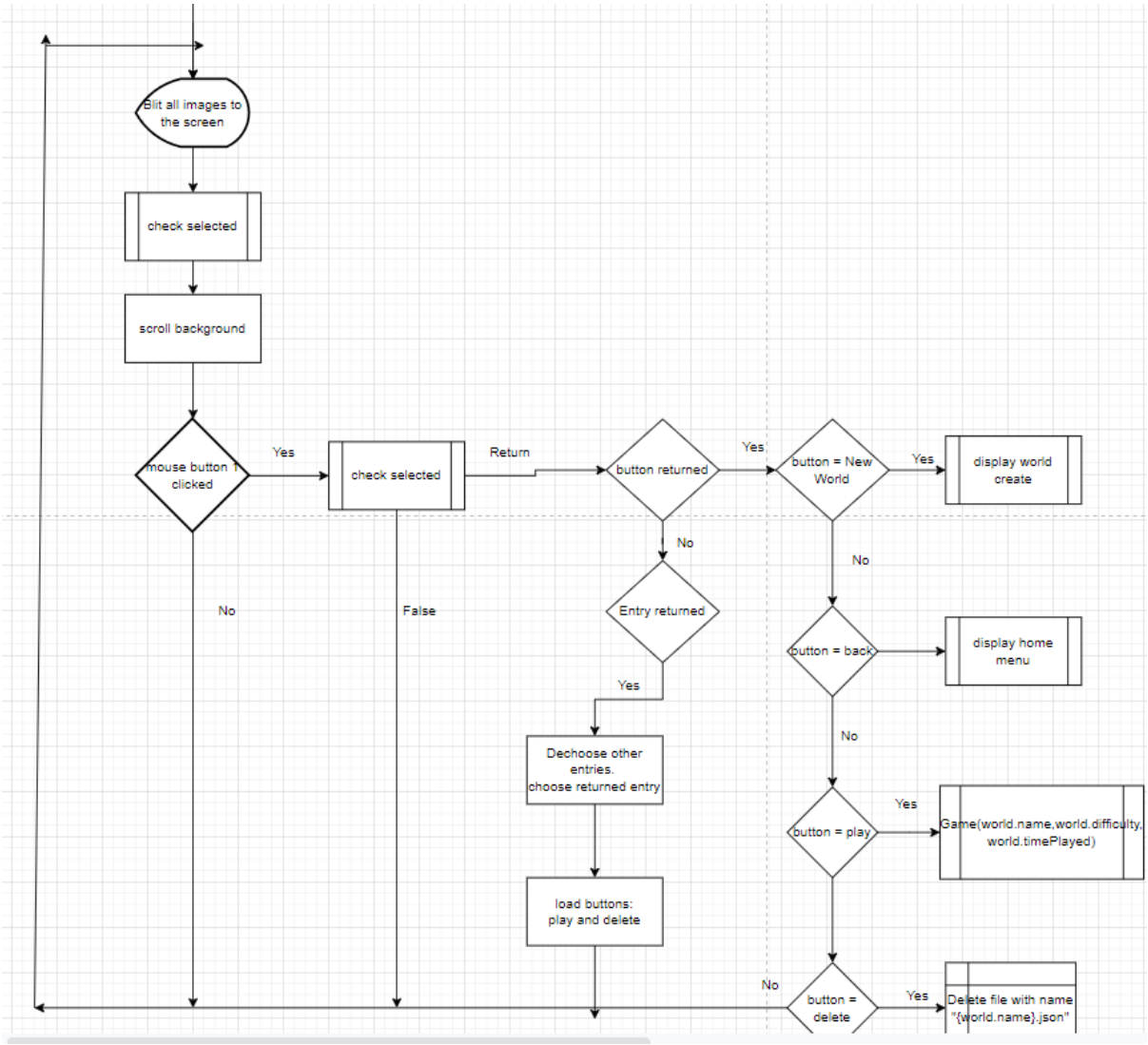


Home menu logic:

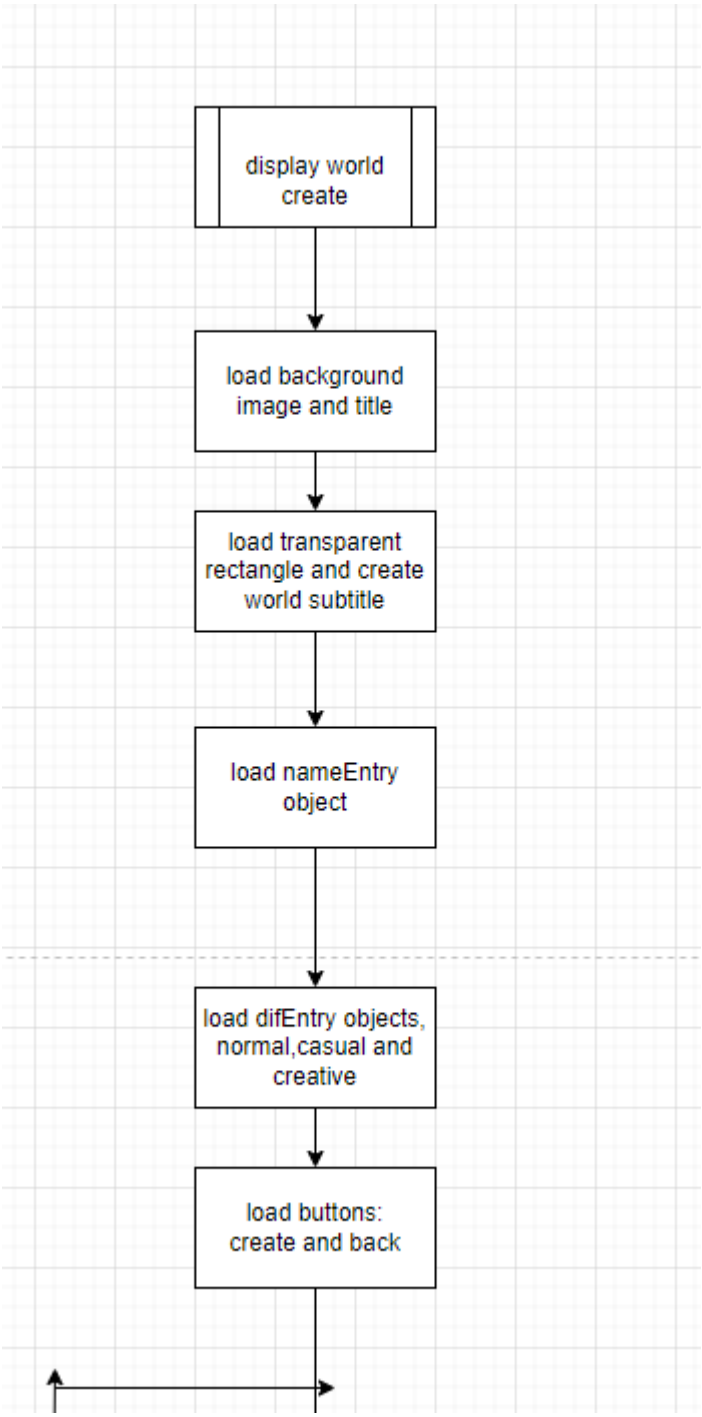


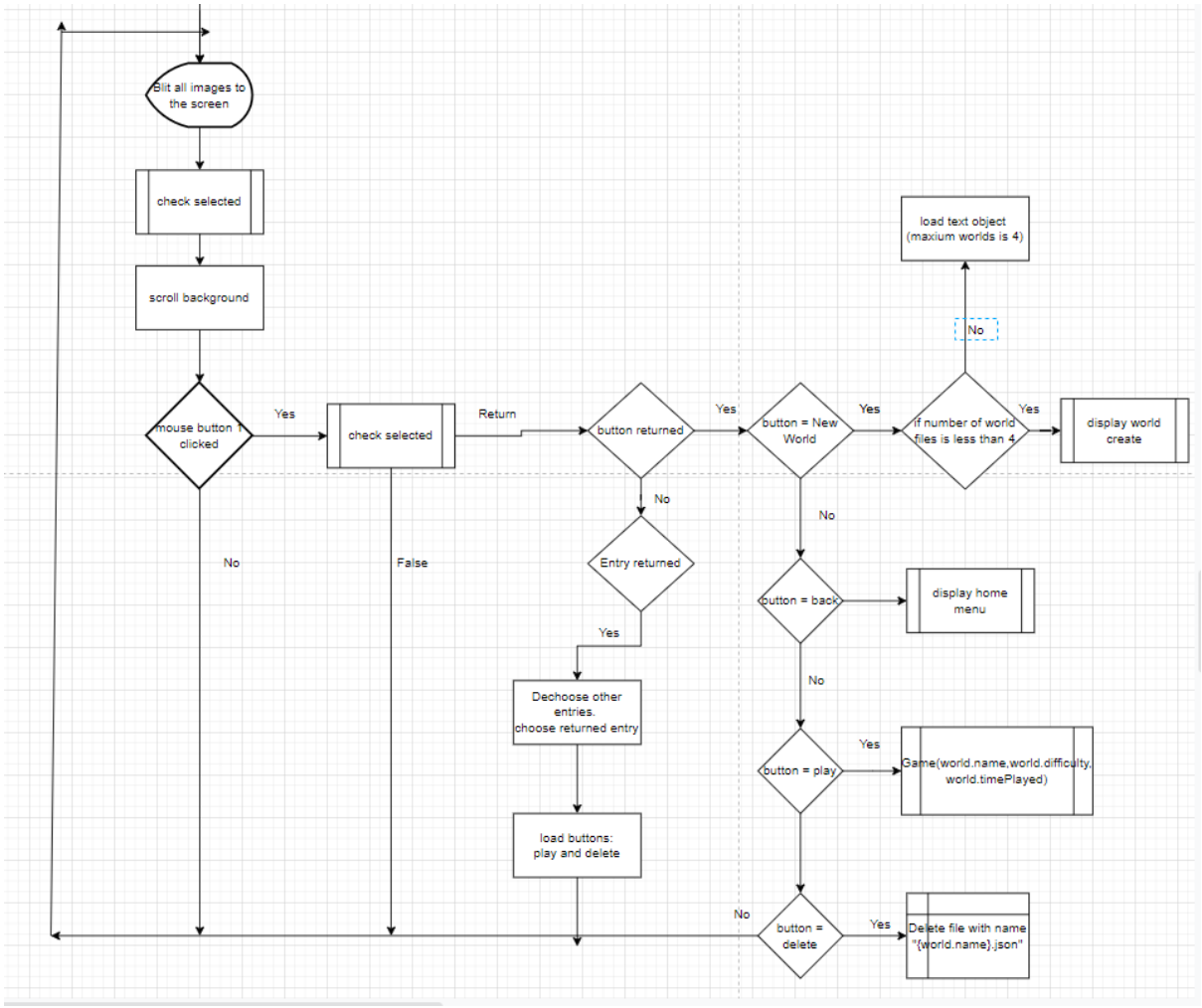
World select logic:






World Create logic:

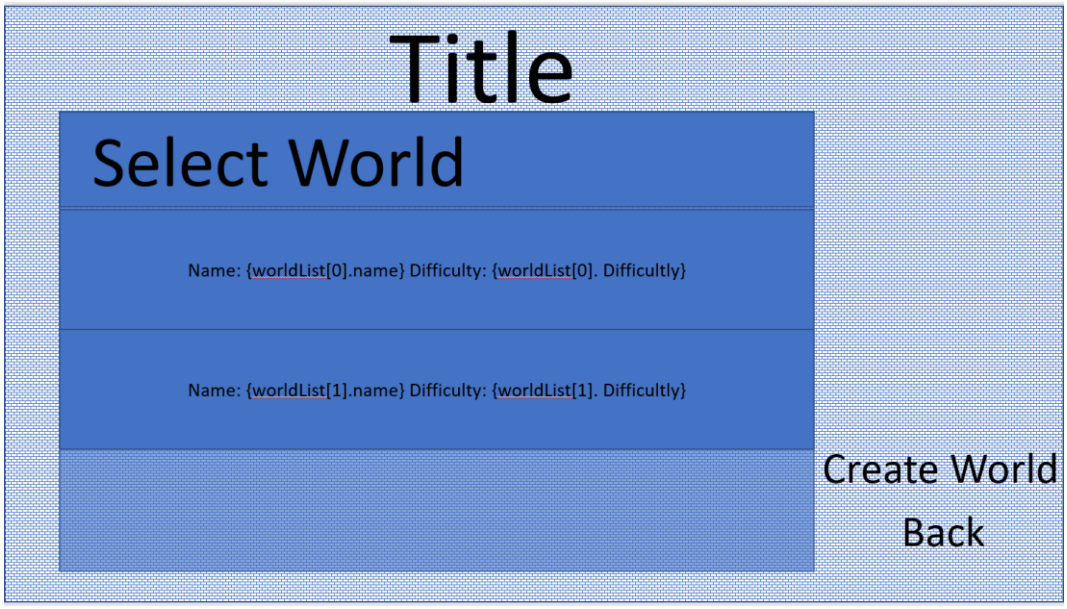

display world create
load background image and title
load transparent rectangle and create world subtitle
load nameEntry object
load difEntry objects, normal,casual and creative
load buttons: create and back


Blit all images to the screen
check selected
scroll background
mouse button 1 clicked
Yes
No
check selected
False
Return
button returned
No
Entry returned
Yes
Dechoose other entries.
choose returned entry
load buttons:
play and delete
Yes
button = New World
Yes
if number of world files is less than 4
Yes
display world create
No
load text object
(maxium worlds is 4)
No
button = back
display home menu
No
button = play
Yes
Game(world.name,world.difficulty,
world.timePlayed)
No
button = delete
Yes
Delete file with name
"{world.name}.json"

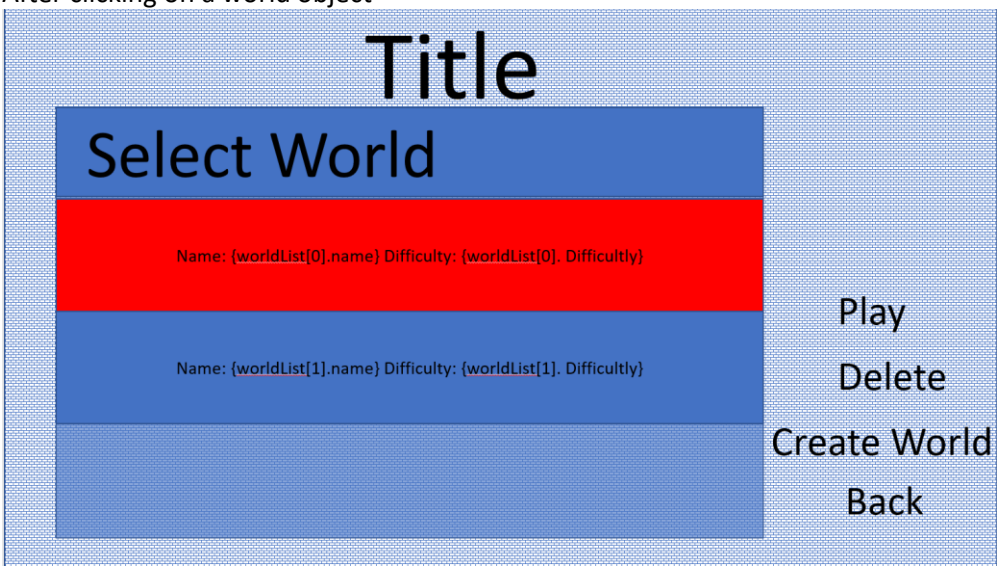
UI:

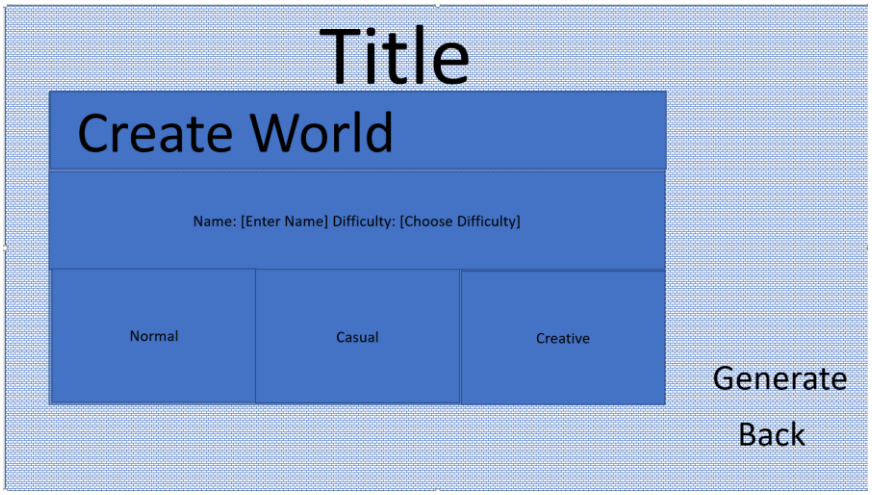

Title
Play
Quit

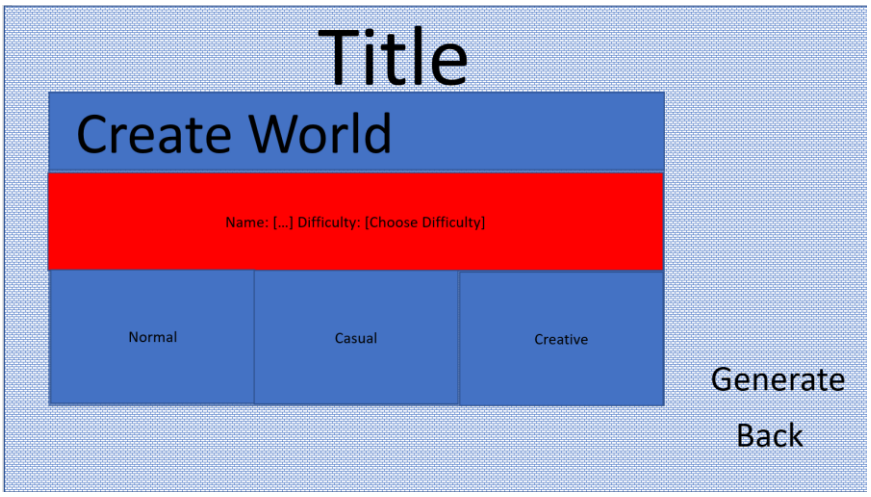
After hovering over a button:

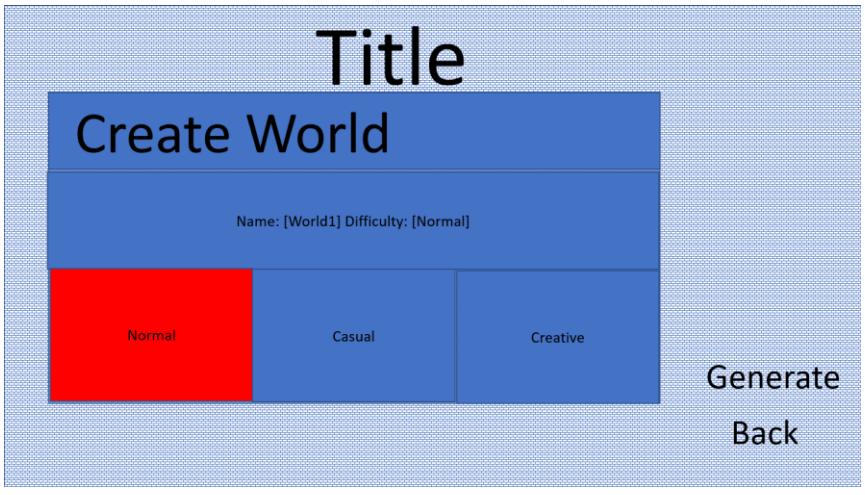

After clicking on a world object






*After selecting the name entry, you should be able to type in the name*




*After selecting the difficulty entry, the name entry difficulty section should change to match it*

## Objective 6) Procedural Generation

## Make world

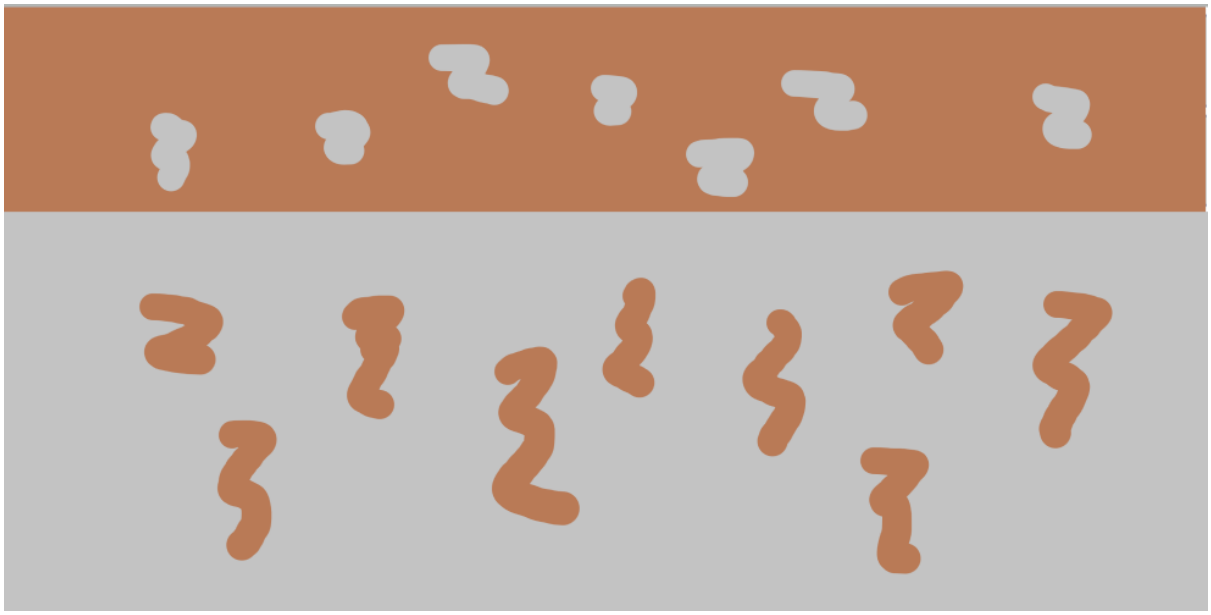
For my research I looked at this video:

https://www.youtube.com/watch?v=Pgt82G4Jxac

Overview:

Step 1)

Create dirt and stone in 1:2 ratio

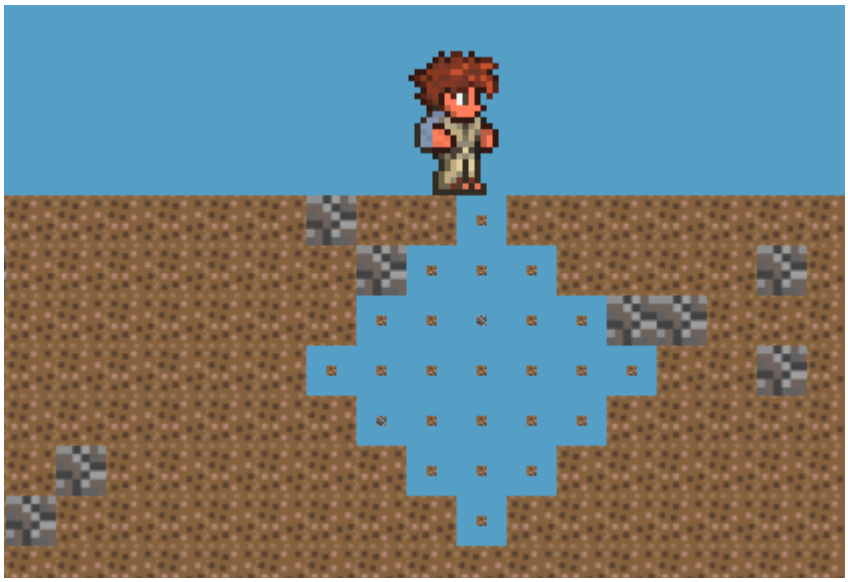


Create long streaks of dirt in the stone section and create short streaks of stone in the dirt section

These streaks are simply a brush algorithm going from side to side at random intervals

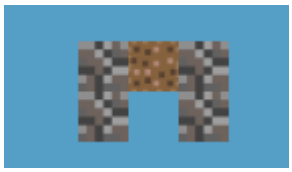
For example it will start at a random world position (x,y) and then choose a random brush size which is the Pythagoras around that world position. For example, brush size 4 then all blocks within x^2+y^2 <= 4 will be coloured to a certain block.

Brush sizes [3,4,5] means for the subroutine to choose a random brush size from this list.



This would be a 4-brush size

Then the subroutine will choose to go to either left or right(50%). Once the direction of chosen, each tile there is a random chance of it choosing the block to the immediate horizontal or diagonal. (50%)



The stone around the dirt block shows the left/leftdown or right/rightdown areas the brush can select.

The brush will do a random number of tiles to the side (width) and then switch to the other side, repeating this a random number of times (depth). The depth for the dirt streaks will be larger than the stone streaks.

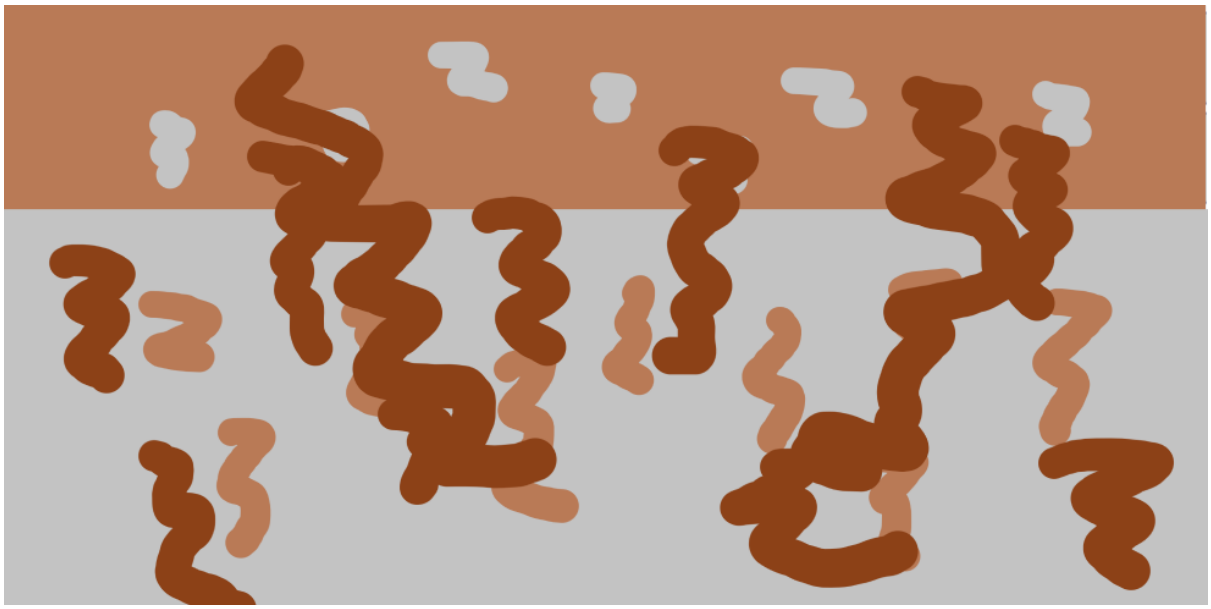
The streak start positions will be in a controlled random distance away from an initial offset.

Let's say initial offset (x,y) then the next one will be ((x+250)+-randint(100), (y+- randint(100)))

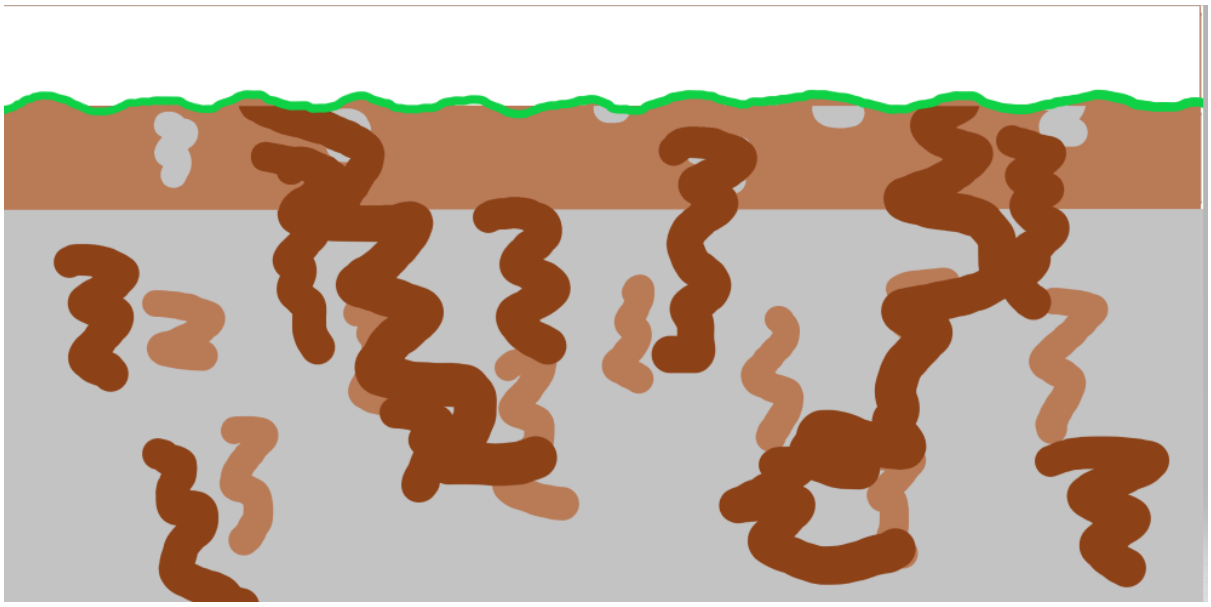
Then the next one from that position would be: ((x+250+250+-randint(100) +-randint(100), (y+-randint(100) +-randint(100)))

Therefore simply be ((newX+250)+-randint(100), (newY+- randint(100))) until the newX position becomes greater than 1000*WORLDLENGTH. The red describes the controlled effect which means that ores are always spaced out from each other. The blue describes the random effect which means that ores will never be found in the same position, adding replay ability into the game.

Then for the caves you simply use the squiggle generation algorithm but with a larger brush and deleting blocks rather than creating them. One difference being is that there will be two major caves that will take the player very far down in the ground.



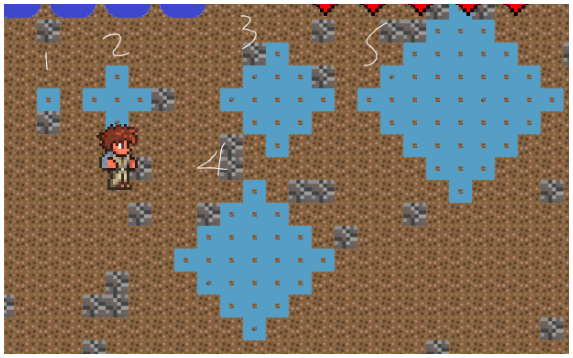
Then your clear for air and delete the tiles on the surface in the form of a sine wave



Fill any air gaps with dirt

Then generate ores based on circles of different shapes and sizes. This means to use the squiggle algorithm but with width and depth that are much smaller and with smaller brushes [1,2,3].

Brush Size




Squiggle pseudo-python code

```
function squiggle (self, BrushRange, numberOnX, numberOnY, positionalRange[X,Y], initialPosition, widthRange[w,r], depthRange[d,r],fill):

    xOffset=((WORLDLENGTH*1000 div TILESIZE) - initPos[0]) div numberOnX
    yOffset=((WORLDHEIGHT*500 div TILESIZE) - initPos[1]) div numberOnY

    SET for x TO 0 to numberOnX:
        SET for y TO 0 to numberOnY:
            startingPos=(initPos div TILESIZE)+((x*xOffset)+randomInt(posRange[0]), (y*yOffset)+randomInt(posRange[1]))
            pos=startingPos

            left IF randomInt(0,1) else right
            SET brush TO randomInt(brushRange)

            SET for height TO 0 to  depthRange[0]+randomInt(depthRange[1])
                SET for width TO 0 to widthRange[0]+randomInt(widthRange[1]):
                    self.brushDraw(pos, brush, fill)

                    side IF randomInt(0,1) else diagonal

                    IF left:
                        SET if side then pos TO pos[0]+1,pos[1] else pos=pos[0]+1,pos[1]+1
                    ELSE:
                        SET if side then pos TO pos[0]-1,pos[1] else pos=pos[0]-1,pos[1]+1

                SET left TO not left
```

*https://pseudocode.deepjain.com/index.php*

Brush Draw pseudo-python code

```
DEFINE FUNCTION brushDraw(self, pos, brush, fill):
rectCorner=(pos[0]-(brush-1), pos[1]-(brush-1))
SET for x TO 0 to (brush*2-1)
    SET for y TO 0 to (brush*2-1)
        testPos=2dVector
        testPos=(rectCorner[0]+x,rectCorner[1]+y)
        IF testPos.magnitude < brush:
            FLOORMAP[testPos]=fill
```

Grass uses multiple sine waves to create a natural looking terrain

## Grass generation algorithm

```
DEFINE FUNCTION drawGrass(self, initPos, sine1, sine2, sine3, sin4, sin5):
    FOR x = initPos[0] to (WORLDLENGTH*1000) div TILESIZE
        y=round(superpose(sin1,sine2,sine3,sine4,sine5)+initPos[1])
        self.brushDraw((x,y), 3, "dirt")
```

### Grass refresh algorithm

If no block above a dirt block then change that block to a grass block

### Darkness algorithm

To add more depth to the game darkness is a necessary feature as there is no fun in a game amount finding materials and ores if the ores are in plain sight. Normally you would use a light path finding algorithm to illuminate the map, not make a darkness algorithm to add dark to the map. However, since pygame is slow doing a path finding algorithm for every source of light would be too intensive for pygame to handle so instead I am using an algorithm that is inspired by the brush algorithm I used earlier to draw the procedural generation.

This algorithm is using different brush sizes to search around a block and see how far you must go until “air” is found (an absence of a block) and then converting this to a transparency value for a surface that we are going to be printing on top of the existing block texture.

First the brush size starts at one and checks all blocks in a dictionary one magnitude away, then at two and checks all blocks 2 blocks away… until it reaches a world position that is not a key for the dictionary and so it is an empty block, an air block and so it can illuminate it.

There is an optimisation that can be applied to this algorithm as when we check a brush size of n we have already checked n-1 and so only the block in magnitude within n-1 and n need to be checked.

Where n is the brush size.

If n-1 < distance.magnitude <= n:

Check

endIf

However, if I left the algorithm just like this then the edges of the world would also be lit up as they have no blocks around them. This not what we want thought and so to fix this we can make it so when the brush finds bedrock (block at the edge of the world) in the sweep it will stop immediately and set the brush size to 5 (max) and so give that area a maximum darkness

However, doing this algorithm for every block is the map would be too heavy for the game and so would result in other parts of the problem breaking done such as collisions and the fps would take a major hit every tile the darkness needs to be updated (objective 16.) This is very important as low fps makes even a fun game boring.

Therefore, I am going to use a dictionary and a per event search, i.e.: the whole map doesn't need to be refreshed on block break, only the surrounding blocks of the broken block.

I am going to search for a max brush size of 5 around the block as that is the range of the brush search.

```
DEFINE FUNCTION updateDarkness(self, block):
    IF block:

        spaceFound=False
        brush=1
        pos=(block.x,block.y)
        WHILE not spaceFound and brush < 5:
            brush+=1
            rectCorner=(pos[0]-(brush-1),pos[1]-(brush-1))
            FOR x = 0 to range(brush*2-1):
                FOR y = 0 to range(brush*2-1):
                    testPos=(rectCorner[0]+x,rectCorner[1]+y)
                    distance=2dVector
                    distance[0]=testPos[0]-pos[0]
                    distance[1]=testPos[1]-pos[1]

                    IF (brush-2) < distance.magnitude() <= (brush-1):
                        IF testPos IN dict:
                            IF dict(testPos).name=="bedrock":
                                brush=5
                        ELSE:
                            spaceFound=True

        block.updateDarkness((brush-2))
```

## Tree generation algorithm

### Logic-

Where there is a grass block spawn a tree with the bottom of its base at the same position as the top of the grass block. If the tree collides with any blocks, then kill it. If the tree collides with any other trees, then kill all the other trees.

Kill trees at random to get a random distribution of trees

### Background generation

In the first step of the make world process, you add the relevant background object to a dictionary at that point; e.g: for the dirt at dirt background and for the stone and stone background. Then once air at the top is cleared remove those dirt background from the dictionary.

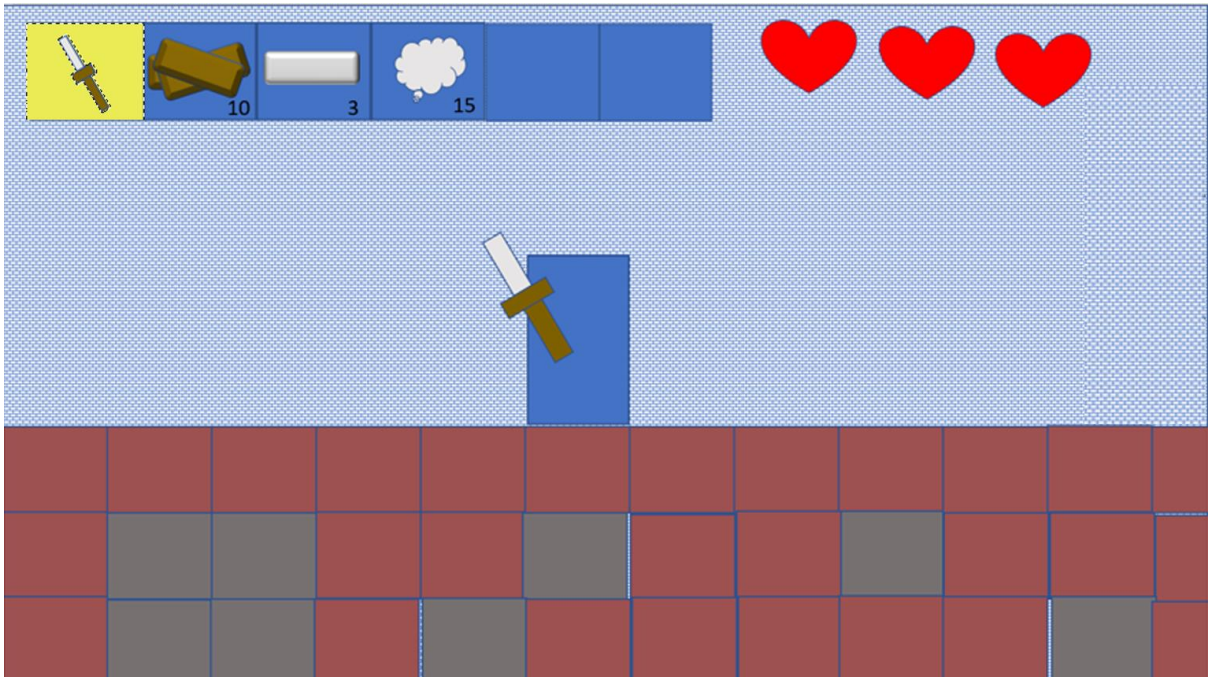
Then you check for all collisions between background sprites and colSprites, if there are collisions then kill the background object. This step is done for optimisation of FPS as blitting sprites of top of each other would result it a very low frame rate. (OB 16)

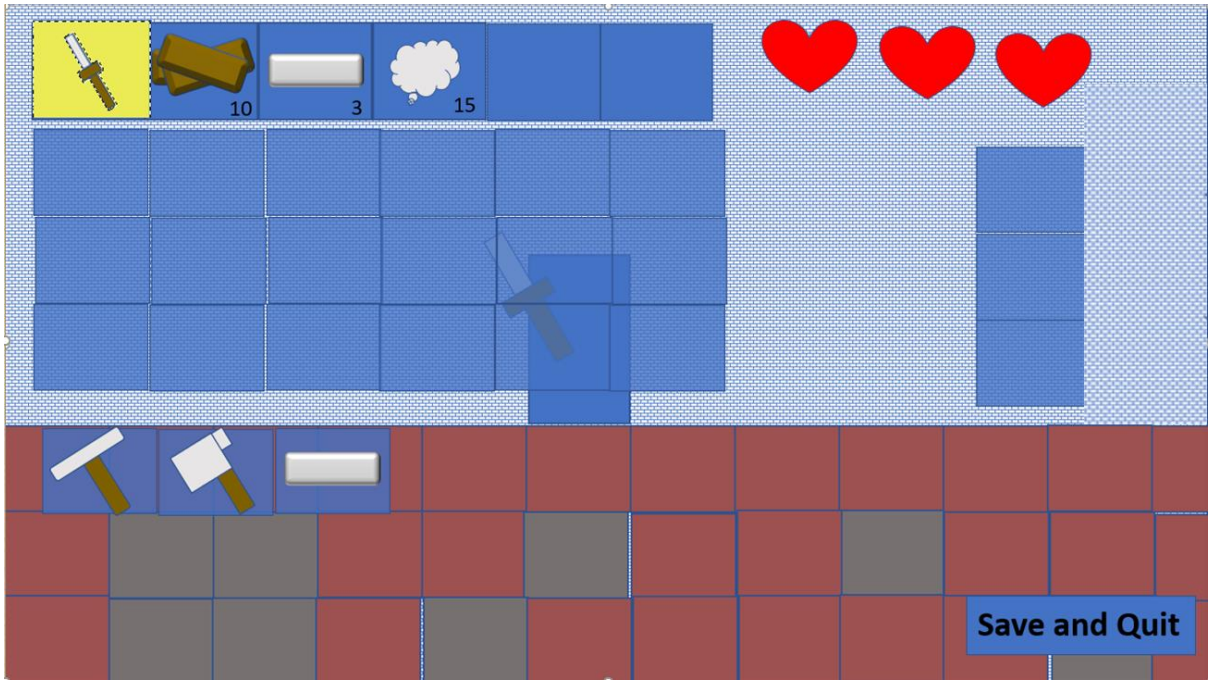
Then when you destroy a block, add the background at the point (if there is one) to the backSprites groups again, this means it is alive again.

## Objective 7) HUD

The HUD or heads up display is arguably the most important thing in a video game as it acts as a guide for the player so that they can get accustomed to the game. A HUD is often a make or break for a game as it is the interface the player with be interacting with the most.

### Modelling:






## Inventory:

*Features of the inventory HUD (displaying inventory)*

1) Contains 3 Sections:

- Hotbar-contains all the tools/weapons/blocks/materials a player needs on quick access
- Armour-slots specifically for wearing armour which allows the player to get damage reduction
- Storage-contains all the other things a player can carry

2) Toggle the storage section of the inventory:

- When ESC is pressed the inventory should show and hide
- If key ESC pressed call toggleInv

3) Scroll to change the selected item in hotbar

- If mouse button up then scroll(True)
- If mouse button down then scroll(False)

4) Use the number keys to change the selected item in hotbar

- changeSelected(numKey)

5)Update the inventory when the player inventory has a change

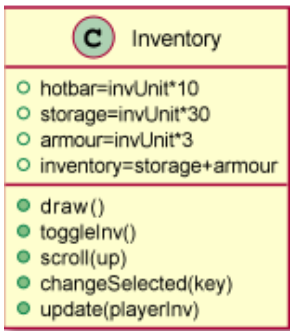
6) Draw each inventory unit, this is done by iterating through the lists of invUnits and then calling draw() on all of them

*Inventory object*

The inventory HUD is going to be a separate object that branches from game, this object is completely different from the player inventory and so when the player inventory is changed it will be passed into the HUD inventory which will then update all the images and numbers.

This inventory object will be using the invUnit object which is simply a singular unit from the inventory.

Class diagram-




*Features of the invUnit:*

1) holds the image for the single inventory unit, base, and selected variant

2)allows status to be changed from base to selected or vice versa

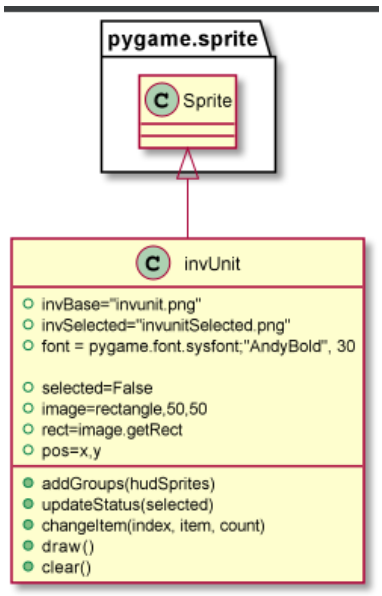
3)you can change the item being held in that unit by referencing the item.index, item.name, item.count

4)The class will draw the unit if the unit is visible (if it is in the hotbar or the inventory is showing), in that unit it will draw the image of the item.name if the count of the item is greater than zero. This will be done by finding the image in the item Dictionary and printing it on top of the inventory unit image. Also, the count of that item also needs to be printed ontop of the invUnit image, this tells a player how much of an item they have and so is very important. The name of item should also appear if the user hovers over it but is not as important as the images do make it clear and so it is more of a quality-of-life feature.

5) be able to stop drawing that object or kill() the sprite which is being printed to the screen, this needs to be done so the player can close the inventory screen.

*Inventory unit object*

Class diagram-




*Functions of inventory:*

### Be able to move items around in the inventory

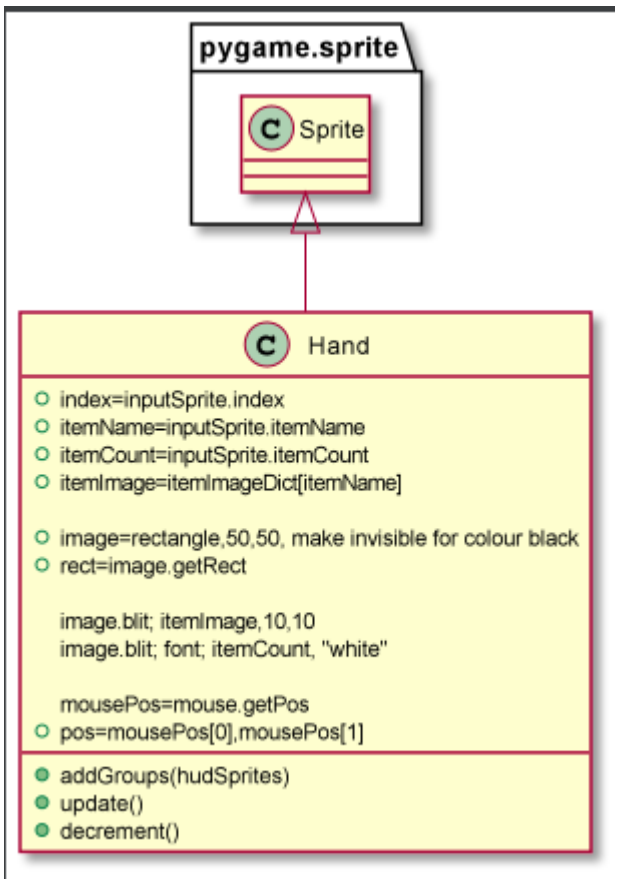
- By clicking on a unit in the inventory the user should be able to move all the contents of that unit into another unit or into a blank unit by using the left click on the mouse.
- Furthermore, by using the right click the user should be able to pick a single item or while having a stack in hand drop a single item at a item into an inventory unit

This can be done by creating a new object called hand which stores the contents of one inventory unit temporarily. Then once there is nothing in hand the object is killed.

This “hand” needs to be able to store a certain item that we can pass as an invUnit sprite into the object initialisation and then drop all of it (kill) or one of it at a time (decrement)

Furthermore, the hand object will need a positional update every frame as we want the contents that we are moving to be stuck to the mouse pointer until we deposit it; therefore, will need an update method.

Hand object class diagram-




Logic for moving items around:

On left click with an invUnit

If both hand and unit empty do nothing

If hand empty and unit filled pick up

If hand filled and unit empty drop off

If hand and unit filled with the same item them try to deposit as much as you can from the hand into the unit. If the unit is already at maxItemCount then swap hand, unit.

If hand and unit filled with a different item name, then swap them.

If clicked off hud then place item back into the invUnit it was in previously

Psuedocode for moving items around the inventory:

```
DEFINE FUNCTION hudEvent(event):

        hudClicked=False

        IF event.button==1:
        // on left click


            FOR sprite IN hudsprites:
                //if there exists a hudsprite at your mouse position
                IF sprite.rect collides with eventPos:


                    //if you click on the inventory
                    IF sprite class name == "invUnit":

                        hudClicked=True


                    //if the unit clicked on is empty then desposit if the hand exists
                    IF sprite.itemName=="empty":

                        IF hand exists:

                            CALL player.invChange(sprite.index, hand.itemName, hand.itemCount)

                            CALL hand.kill()

                            hand=None


                    //otherwise
                    ELSE:
                        //if hand exists
                        IF hand exists:

                            //try to combine hand and invUnit

                            IF hand.itemName==sprite.itemName:

                                IF player.invChange(sprite.index, hand.itemName, hand.itemCount+sprite.itemCount):

                                    CALL hand.kill()

                                    hand=None

                                ELSE:
                                    //if the change didn't go through because total item count of both hand and unit exceeded the maxItemCount

                                    IF sprite.itemCount==32:
                                        //if the invUnit contains the max number of items, swap hand and invUnit.
                                       CALL player.invChange(sprite.index, hand.itemName, hand.itemCount)

                                        CALL hand.kill()

                                        hand=Hand(sprite)
                                        //otherwise
```

```
                                        hand=Hand(sprite)
                                        //otherwise
                                    ELSE:
                                        //fill the rest of the invUnit with that item and then keep the left over in the hand
                                        sprite.itemCount=hand.itemCount+sprite.itemCount - player.maxItemCount

                                        player.invChange(sprite.index, hand.itemName, player.maxItemCount)

                                        hand.kill()

                                        hand=Hand(sprite)



                                ELSE:
                                    //swap hand and invUnit because the two items are different
                                    player.invChange(sprite.index, hand.itemName, hand.itemCount)

                                    hand.kill()

                                    hand=Hand(sprite)



                            ELSE:
                                //pull invUnit into Hand

                                player.invChange(sprite.index,"empty",0)

                                hand=Hand(sprite)

            //drop the hand where it was picked from if you click outside of the inventory
            IF hudClicked==False and hand:

                intialCount=player.inventory[hand.index][2]

                player.invChange(hand.index,hand.itemName,intialCount+hand.itemCount)

                hand.kill()

                hand=None

                hudClicked=True
```

Right click logic-

If hand empty and unit empty do nothing

If hand filled and unit empty, then drop off 1 item at a time by using the decrement fuction on the hand

If hand empty and unit filled, then pick up one item from the unit

If the hand and unit and both filled but with the same item, still try to drop off one at a time, if not possible then swap items.

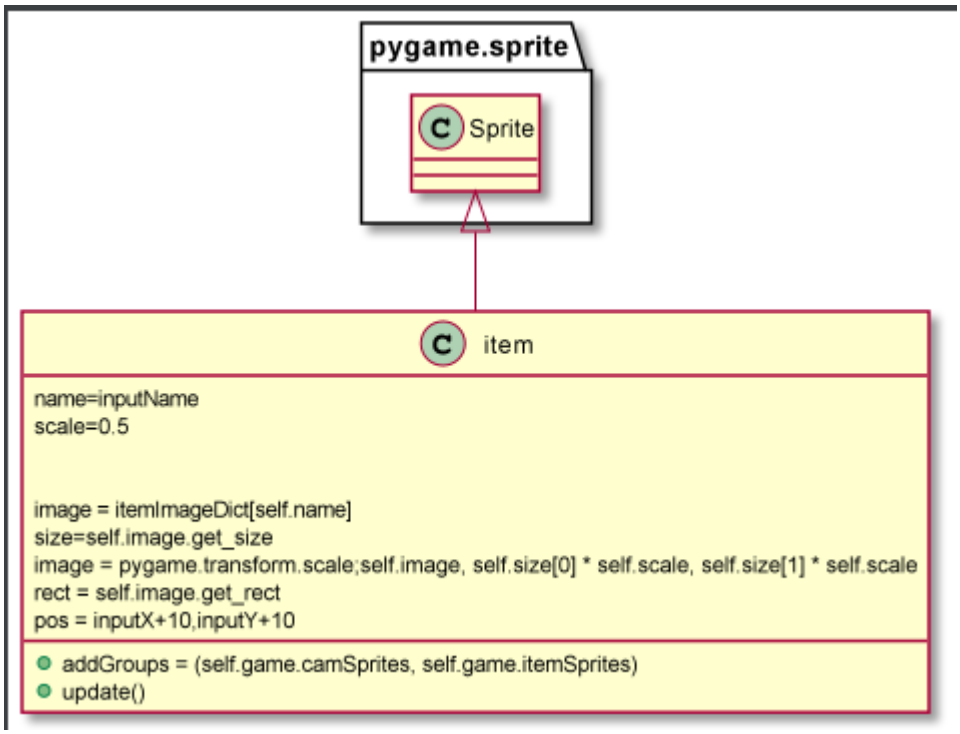
If hand and unit both filled with different items than swap

If clicked off hud then drop item

## Be able to throw items out of the inventory:

Do to this we can create a new class called “item” which will be the dropped state of every item in tile/block/wall/weapon/tool/material in the game.

This is simply a smaller version of the original sprite by using the pygame.scale.transfom function and then spawning a smaller version at the mouse position of where it was dropped. This item also needs an update function so that we can delete it after some time has passed. In the update we can decrement a timer which when gets to zero the item will kill. This is necessary for the player to be able to bin items and to optimise FPS (objective 16)




To do this I am going to utilise the right click when holding an item in hand.

```
IF hudClicked is False and hand exists:
    FOR item = 0 to (hand.itemCount):
        Item(hand.rect.x-camOff[0],hand.rect.y-camOff[1],hand.itemName)
    CALL hand.kill()
    hand=None
    hudClicked is True
```

## Health bar:

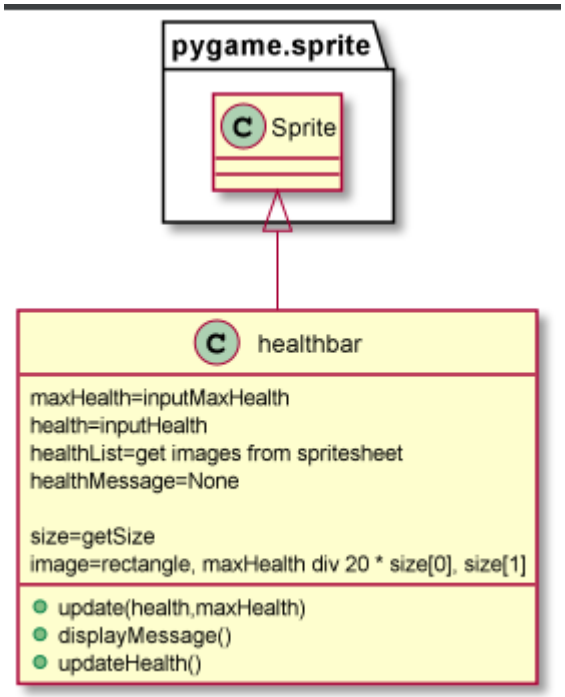
### *Features of health bar HUD (display)*

1) Be able to display health in the form of a user-friendly key, utilising one heart as 20hp and a fraction of a heart as a different shade of red.

2) To be able to update the HUD health bar periodically by passing the value of the player health in whenever there is a change to the player health. – To do this efficiently we need an algorithm that will calculate what hearts we need to print to the screen.

3) The user should be able to tell exactly what health they are on by hovering over the health bar; this will display a message

### *Health bar object:*




displayMessage logic –

if mouse pointer colliding with the rectangle of the health bar, display health at the mousePos + (10,10)

updateHealth pseudocode-

```
DEFINE FUNCTION updateHealth():
    tempHealth=health

    FOR i = 0 to maxHealth div 20:
        IF tempHealth>=20:
            heart=20
            tempHealth-=20
        ELSEIF tempHealth>0:
            heart=tempHealth
            tempHealth=0
        ELSE:
            heart=0

        print onto health bar image - healthList[heart]
```

## Objective 8) Player inventory and health

The HUD is simply the display for these aspects of the player class:

### Player inventory:

The player inventory is a 2d list of 3 values per index position in the list= [index,name,count]

#### *Data Structure:*

It may seem unnecessary to store the index along with the item in the list, however, this saves us a lot of computation time as we need to pass these 3 parameters into the change inventory for the hud and so saves us trying to find the index every time. This is because the invUnit requires the index position argument as otherwise it has no way of knowing what index the item is being stored in. This is needed for example when returning at item to its place in the inventory as you need to return it to its previous index.

#### *Functions of the player inventory:*

The player needs to be able to use the inventory to be able to:

- pick up items
- add items to the inventory through other means such as crafting
- Change what is in the inventory at a specific position, for example after placing down a block the inventory needs to be changed.

For each of these functions we can create a new method under the player class:

### 1) Item pick

Logic-

If player collides with an object in sprite group “itemSprites” then kill that object and call addToInv for that item. So (“item.name”, 1)

## 2) addToInv

This is a function that sweeps the player inventory for open spaces so that several items can be added to it at the nearest instance

Pseudocode-

```
DEFINE FUNCTION addToInv(name, count):
    found=False
    FOR unit IN self.inventory:
        IF unit[1]==name and unit[2] < maxItemCount and unit[0] < 40:

            count=unit[2]+count

            IF count > maxItemCount:
                CALL invChange(unit[0], name, self.maxItemCount)
                count-=self.maxItemCount
                CALL addToInv(name,count)

            ELSE:
                CALL invChange(unit[0], name, count)
                count=0

            found=True
            break

    IF not found:
        FOR unit IN self.inventory:
            IF unit[2]==0 and unit[0] < 40:

                IF count > maxItemCount:
                    CALL invChange(unit[0], name, self.maxItemCount)
                    count-=maxItemCount
                    CALL addToInv(name,count)

                ELSE:
```

```
        ELSE:
            CALL invChange(unit[0], name, count)
            count=0

        found=True
        break

IF not found:
    FOR unit IN self.inventory:
        IF unit[2]==0 and unit[0] < 40:

            IF count > maxItemCount:
                CALL invChange(unit[0], name, self.maxItemCount)
                count-=maxItemCount
                CALL addToInv(name,count)

            ELSE:
                CALL invChange(unit[0], name, count)
                count=0


            found=True
            break


IF not found:
    FOR i = 0 to count:
        Create Item(playerX div TILESIZE, playerY div TILESIZE-3 ,count)

    count=0
```

As we can see to use this function, we also need to call invChange.

### 3) invChange

This is simply a function that is used to change a single index of the player inventory.

Logic-

Pass in the 3 parameters: index, name, count

If the count > maxitemCount then the change is not possible return false

If the count is less than 1 after the change the inventory index should be changed to an empty unit [index, “empty”, 0]

Otherwise, do the change.

Flag a change has been made so the HUD knows to refresh the inventory

### Player health:

Player health is simply an integer variable however taking damage is not as simple as just removing health as a player will have a regen cooldown applied to it and timer which limits how much a player can regen.

To implement player health properly we will need 3 functions:

1) updateHealth, this is the update function which is called by the player class to regen health and to check if the health is zero to kill the player. In this function the timers are decremented. The call to heal(amount) to regen health will only take place when both timers are zero. After the call to the regen, the timer will be set to x number of frames so that player regen is not constant. This helps with objective 16 (optimisation) as changing the health and so the HUD every frame would be very heavy on the processor.

2) takeDamage(damage), this is the function that's removes health from the player inventory and start the regeneration cooldown (set to x number of frames) as it wouldn't make sense for the player to start regenerating health immediately. This function will also need to contain elements of objective 15 (Difficulty) as depending on the game mode you will take damage differently. On creative damage will simply not be dealt while on casual the damage is halved. After the health is changed healthChange is flagged so the HUD can update

3) heal(amount), will add an amount of health to the player health making sure that player health doesn't go over the cap. Also flagging healthChange if a change has been made.

## Objective 9) Items

In this objective I will be adding tools and weapons to the game which can be swung, swinging is dual purpose, and it will deal damage to enemies who are intersecting with the mask of the tool/weapon image and if it is a tool that does something useful then it will also perform that action on use.

### Fetching image for each tool

We can fetch a separate image for each tool by using a spritesheet and loading each image out from it like we did with the animations. This time, the image can be added to a dictionary in such a way that calling the item name will fetch the image.

### Implementing tool object.

To implement the tool, we need to model it as an object and give it a mask for hitting enemies, an update method for changing the image per frame and a collision method for checking collision with the enemy. Furthermore, it will need different attributes so that the strength of different tools can be decided:

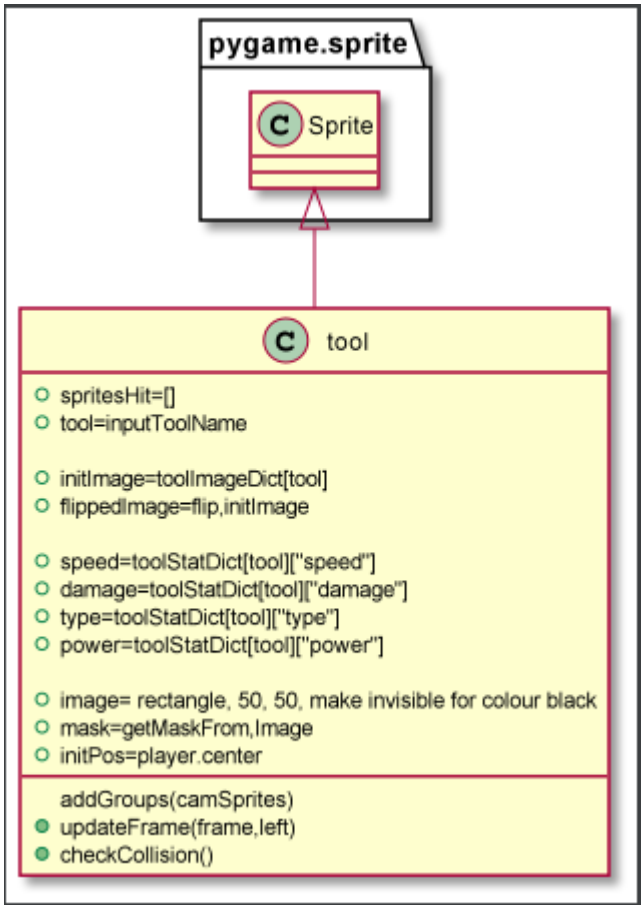
Tool.speed – how fast an animation with that tool is carried out

Tool.damage – how much damage it does per hit

Tool.power – how effective of a tool is it (e.g: will take less hits to mine a block)

Tool.type – what type of block can it destroy

Tool object class diagram:




camSprites is the camera group and so it allows it to stick to the player

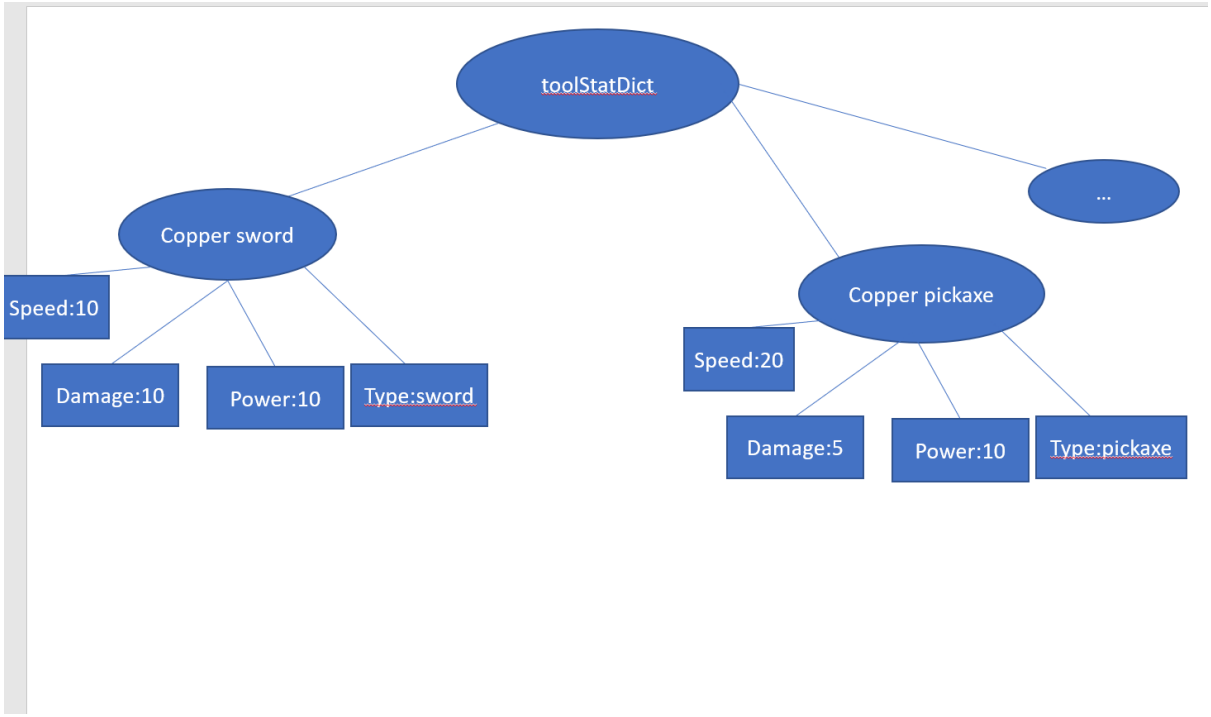
### *Storing data in main memory for tool*

Since we need to store the tool name, speed, damage, power, type for every tool in the game which would be about 21 tools, doing this directly in python would be extremely inefficient. That's where we can use a json file. JSON format will allow us to store a large python dictionary outside of the main program and simply call the name and attribute of tools.

https://www.youtube.com/watch?v=9N6a-VLBa2I

I used this video to figure out how to load from json files.

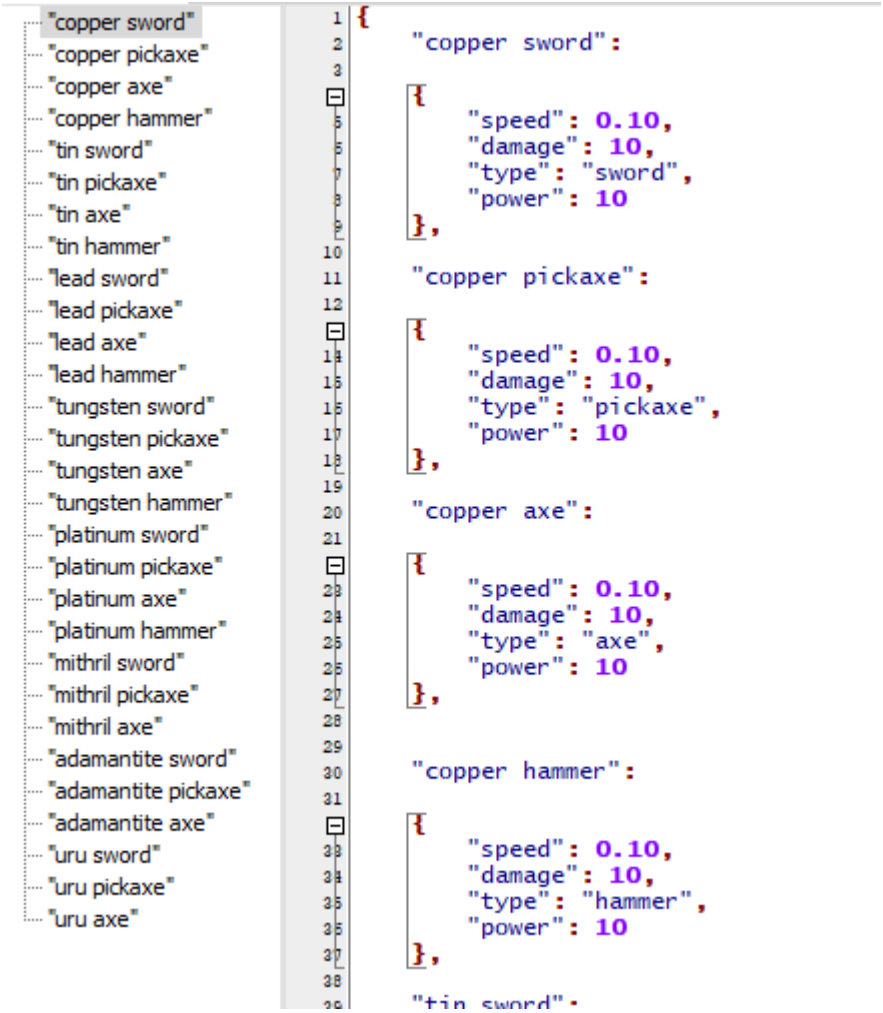
Model for json file:




The ellipse is a dictionary

The rectangles are individual key:elements in the dictionary.

… represents many more tools

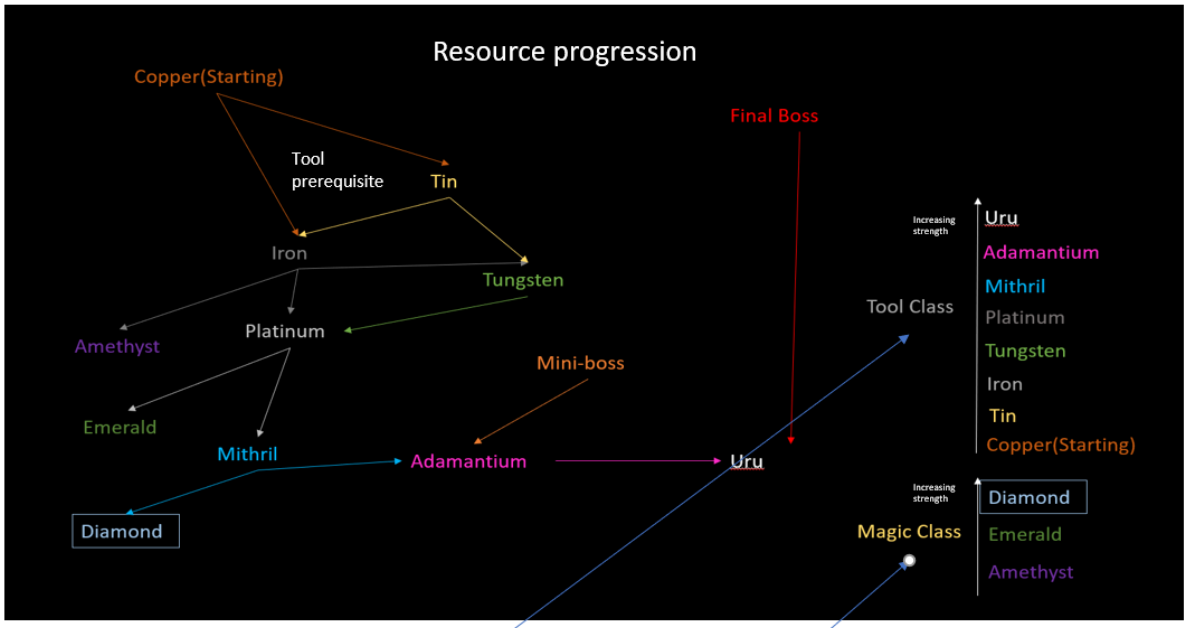



I added all the tools into the Json file

I used this website to check for valid json file.

https://jsonlint.com/

*Progression of tool strength*




This is showing the progression of tool strength.

The arrows show that you need to obtain that material.
For example: with a copper pickaxe you can mine iron and tin.
To obtain adamantium you need mithril tools and to have killed the mini-boss

All tools can mine the ores below their strength level

This is showing the progression of magic item strength

According to this chart I made in the analysis, I will need to adjust tool strengths accordingly so they only mine what they can.

Since with every hit I am going to be doing damage to the health of the block, I am going to make it so that if a pickaxe takes more than 5 hits to destroy a block then it is unable to even damage a block.

Therefore, each tool can just about mine the block it needs.

Pickaxe strengths:

Copper: 10

Tin: 15

Lead: 20

Tungsten: 25

Platinum: 30

Mithril: 40

Adamantite: 70

Uru: 170

Block Strengths:
Dirt: 20

Grass: 20

Stone: 30

Tin: 40

Lead: 50

Tungsten: 70

Platinum: 90

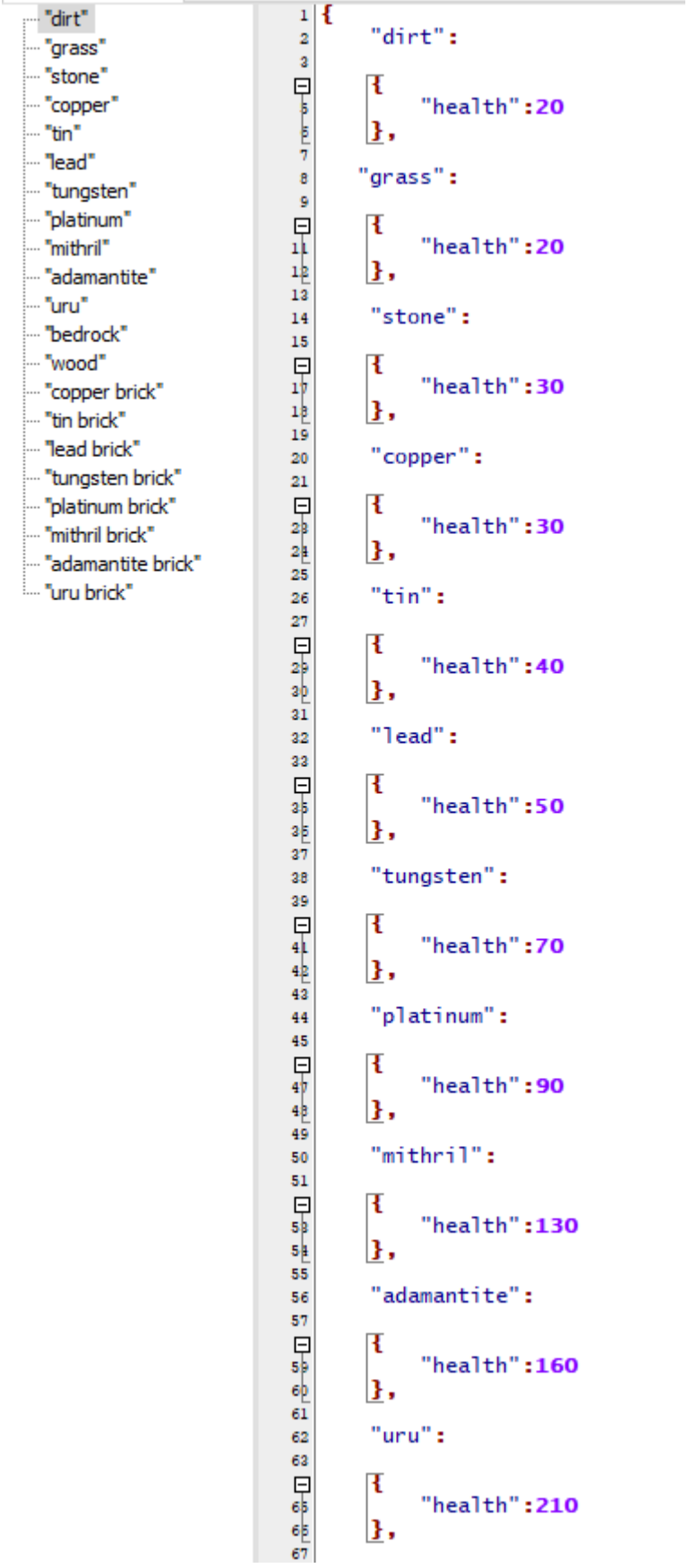
Mithril: 130

Adamantite: 160

Uru: 210

Bedrock: 1000 (therefore unbreakable)

Since blocks also have their own health value, I am going to make a json file for the health values for all blocks as well.


"dirt"
"grass"
"stone"
"copper"
"tin"
"lead"
"tungsten"
"platinum"
"mithril"
"adamantite"
"uru"
"bedrock"
"wood"
"copper brick"
"tin brick"
"lead brick"
"tungsten brick"
"platinum brick"
"mithril brick"
"adamantite brick"
"uru brick"
{
"dirt":
{
"health":20
},
"grass":
{
"health":20
},
"stone":
{
"health":30
},
"copper":
{
"health":30
},
"tin":
{
"health":40
},
"lead":
{
"health":50
},
"tungsten":
{
"health":70
},
"platinum":
{
"health":90
},
"mithril":
{
"health":130
},
"adamantite":
{
"health":160
},
"uru":
{
"health":210
},

Only ores have a large health bar, the brick variants for that block are made easily breakable as it would be annoying in building to have to hit a block as much as 5 times to break it, whereas in resource gathering it makes sense as ores are meant to be tough to mine.

All walls and tiles will have a health of 30

This means walls and tiles on average will be able to be one hit by a decent pickaxe

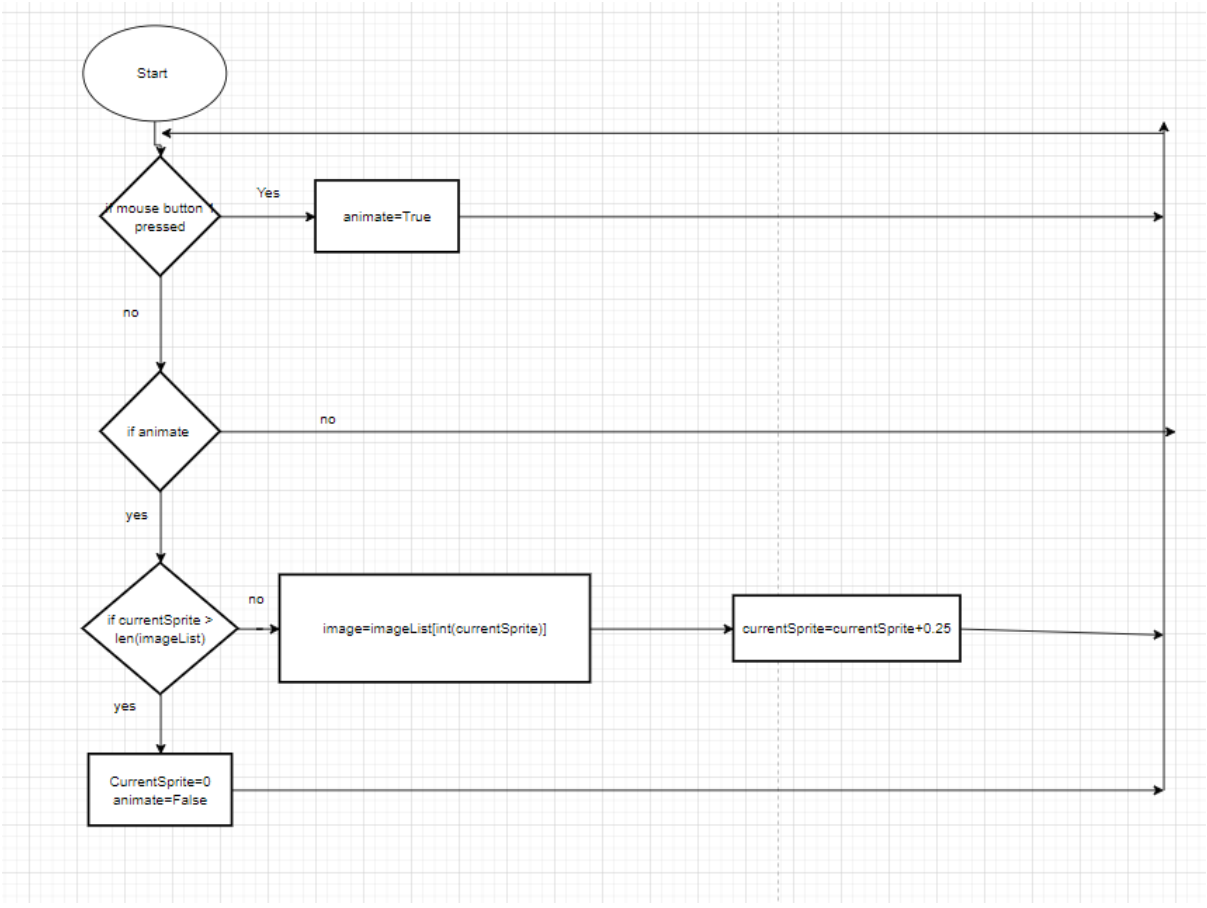
All trees will have a health of 100

This means that trees will take 10 hits to destroy, going down by one hit each time as tool strength improves. Quickest is 3 hits. I don't want to make trees too easy to destroy as it is simple and easy to find trees and so to balance that out the wait to obtain must be longer.

### *Implementing the animation*

To implement the swing, I will need to also work on animations (objective 14), this is because the animation finish will act as a timer for the time taken to do a certain action e.g mine a block.

For a basic animation flow chart view objective 14-




For this animation all we need to do is when the player is doing its using animation we create a tool object and then glue it to the player animation per frame the player is on by using the updateFrame method (4 frames.) However, that Is not enough, the tool must be glued onto the animation when the player is facing both left and right and so the two image attributes are needed in the tool class (that is what the "left" parameter is for). Although this is a bit fiddley as it requires you to get the

position perfect for 8 different situations.  It seems like it is the easiest way to do it as all tools fit within a 50x50 rect and so no repositioning will be needed. The only other alternative would be to make a new spritesheet for every single tool and that is a lot more work than this.

### *Carrying out a use with tool*

Useful tools include:

- Pickaxe, then on use it will check for a block at the mouse position and then will deal damage to that block.
- Axe: on use it will check for a tree at the mouse position and then will deal damage to the tree
- Hammer: on use it will check for a wall or tile at the mouse position and then will deal damage to the wall/tile.
- Hamaxe: combination of hammer and axe

When animation = False we can call a finishUse subroutine which will then carry out the action for the tool.

Initial pseudocode for using a tool.

```
DEFINE FUNCTION finishUse(self):
    IF tool exists:
        IF tool.type=="pickaxe":
            destroyBlock(self.event, self.tool)
        ELSEIF tool.type=="axe":
           destroyTree(self.event, self.tool)
        ELSEIF tool.type=="hammer":
            destroyWall(self.event, self.tool)
        ELSEIF tool.type=="hamaxe":
            destroyTree(self.event, self.tool)
            destroyWall(self.event, self.tool)

    ELSE:

        self.game.place(self.event)
```

The game class is handling the breaking of blocks/walls/tiles.

### *Dealing Damage with tool*

Every tool also needs to do damage on swing otherwise the game won't make sense.

To do this we can get a new mask every time the updateFrame fuction is called.

Then we can use the checkCollision method in the tool object to check for a mask collision between the tool and any sprite in enemySprites (sprite group). If there is a collision the tool deals damage (tool.damage attribute is the measure for how much damage a tool will do to an enemy) and then adds the sprites to a hitlist and so the same sprite cannot be hit twice during the lifetime of the tool (1 swing)

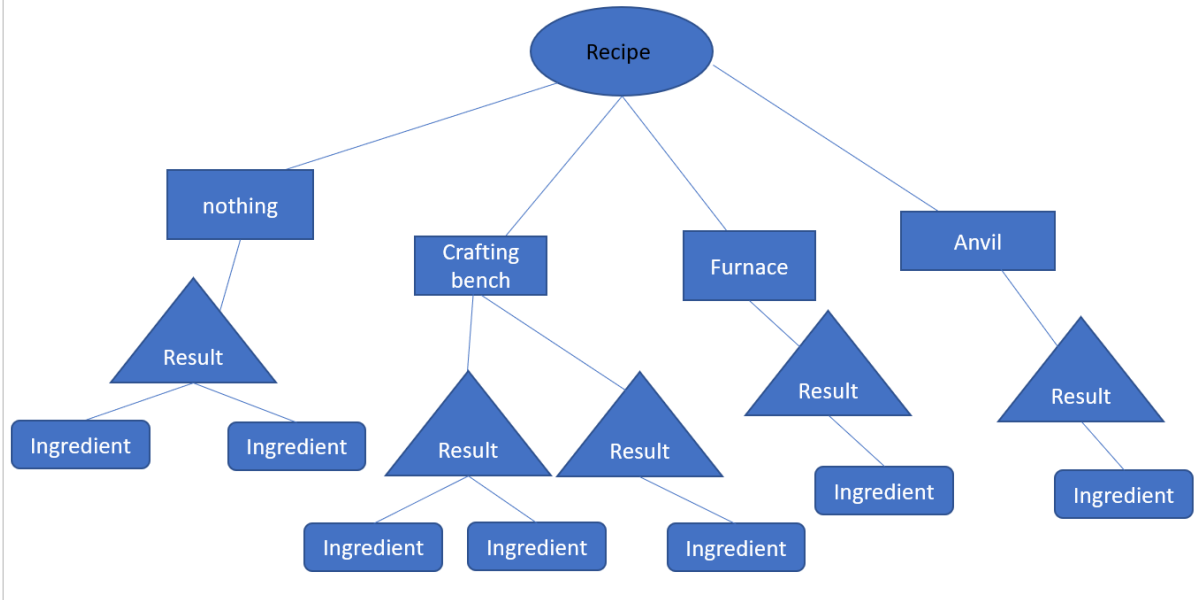
## Objective 10) Crafting:

I am now working on objective 10 which is crafting.

Outline:

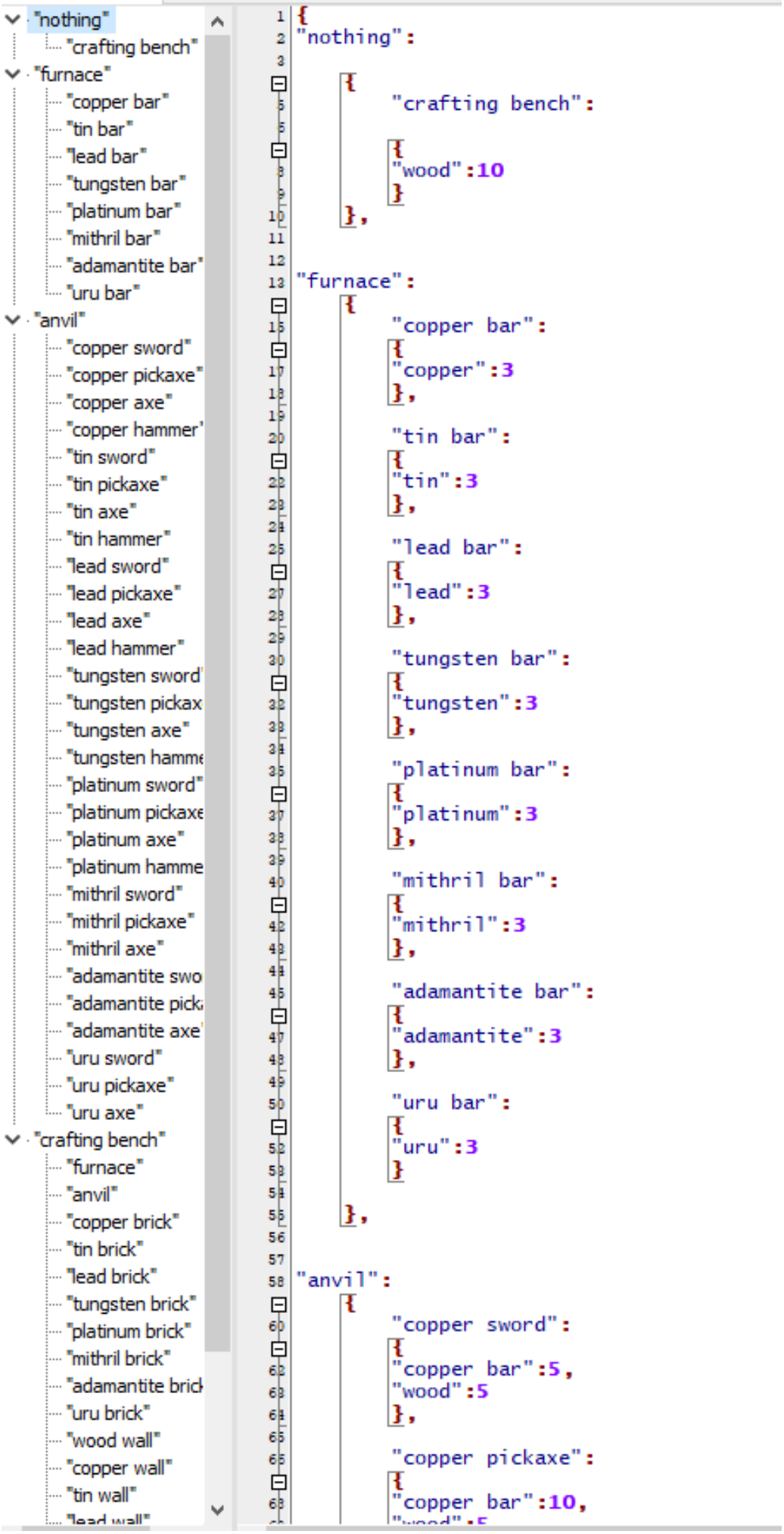
When you open the inventory the things you can craft is displayed based on the materials you have in your inventory and whether you are colliding with of the of the crafting tiles: ("crafting bench", "furnance", "anvil") if you are not colliding with any of these then you will simply be allowed to make a crafting bench.

I am going to do this by using a dictionary stored in a json file, this will store the recipies like this:



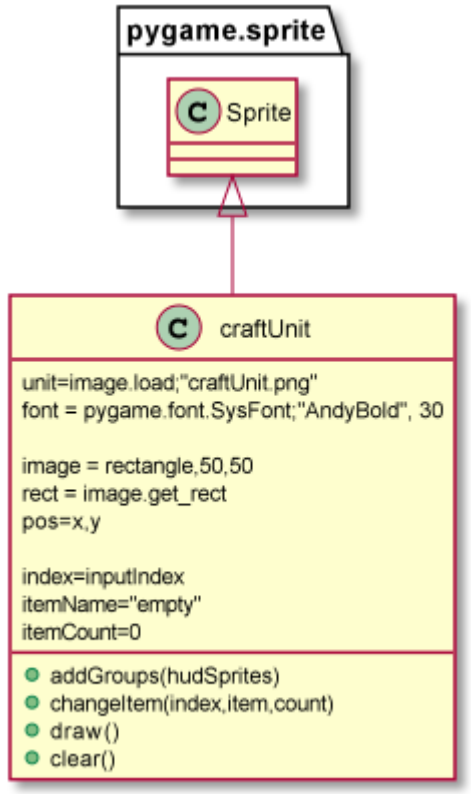

All of the shapes in this image a python dictionaries, apart from the final leaves of this tree as they are the individual key:element pairs for each result in the dictionaries.

I wrote out the recipes into a json file:


"nothing"
"crafting bench"
"furnace"
"copper bar"
"tin bar"
"lead bar"
"tungsten bar"
"platinum bar"
"mithril bar"
"adamantite bar"
"uru bar"
"anvil"
"copper sword"
"copper pickaxe"
"copper axe"
"copper hammer"
"tin sword"
"tin pickaxe"
"tin axe"
"tin hammer"
"lead sword"
"lead pickaxe"
"lead axe"
"lead hammer"
"tungsten sword"
"tungsten pickax
"tungsten axe"
"tungsten hamme
"platinum sword"
"platinum pickaxe
"platinum axe"
"platinum hamme
"mithril sword"
"mithril pickaxe"
"mithril axe"
"adamantite swo
"adamantite pick
"adamantite axe
"uru sword"
"uru pickaxe"
"uru axe"
"crafting bench"
"furnace"
"anvil"
"copper brick"
"tin brick"
"lead brick"
"tungsten brick"
"platinum brick"
"mithril brick"
"adamantite brick
"uru brick"
"wood wall"
"copper wall"
"tin wall"
"lead wall"
{
"nothing":
{
"crafting bench":
{
"wood":10
}
},
"furnace":
{
"copper bar":
{
"copper":3
},
"tin bar":
{
"tin":3
},
"lead bar":
{
"lead":3
},
"tungsten bar":
{
"tungsten":3
},
"platinum bar":
{
"platinum":3
},
"mithril bar":
{
"mithril":3
},
"adamantite bar":
{
"adamantite":3
},
"uru bar":
{
"uru":3
}
},
"anvil":
{
"copper sword":
{
"copper bar":5,
"wood":5
},
"copper pickaxe":
{
"copper bar":10,

To display the new crafting GUI, we will need:

A new craftUnit class which is identical to invUnit but it is of a red colour and doesn't have the all the functions of the invUnit needs to be created. This is because the difference in the crafting and inventory menu needs to be distinct to the user. Furthermore, the invUnit carries the behaviours of the inventory and couldn't be used for crafting.




Lost methods – change selected

Lost attributes – selected Image

A new list to the inventory class which stores all the craftingUnits. In which, the craftingUnits display the recipes available to the player.

CraftingList = CraftUnit*30

A new subroutine to the inventory class which would check what tile a player is touching and check crafting recipes from the dictionary accordingly. If the player had the materials necessary to craft a resulting item, then that recipe will be passed into another subroutine which finds the next empty slot in the crafting recipe list in the inventory class. I will call this method "updateCrafting"

Also, a second method is needed to actually add the recipe onto the crafting unit – I will call this method addRecipe(name,amountYouCanCraft)

A third method is needed to clear the crafting after the inventory has been closed so that the crafting recipies can be added once more when the inventory is opened.

Pseudocode for update crafting:

```
DEFINE FUNCTION updateCrafting(self):
    CALL clearCrafting()


    IF player.touching:
        tile=player.touching


    ELSE:
        tile="nothing"


    FOR key IN recipeDict[tile]:
        ingList=list(recipeDict[tile][key].keys())
        itemCounts=[]
        FOR ing IN ingList:
            materialCount=0
            FOR unit IN player.inventory:
                IF ing==unit[1]:
                    materialCount+=unit[2]


            itemCounts.add(materialCount div recipeDict[tile][key][ing])

        IF difficulty !="Creative":
            numItems=min(itemCounts)
            IF numItems>=1:
                addRecipe(key,numItems)
        ELSE:
            addRecipe(key,infinite)
```

Logic for method – “addRecipe”

Check for each unit in the craftingList and if it is empty then change the craftingList[index] item to the name of the recipe and number of items craft able into that index; this is by using the changeItem method of craftUnit.

Logic for method – “clear crafting”

Iterate through the craftingList changing every index in the list to [index, “empty”, 0]

The hud event subroutine needs to handle clicks onto “craftUnit” class and for the recipe you clicked on remove the ingredients needed from your inventory and then add that item to the inventory in the next available place.

To do this effectively we can use the previously mentioned addToInv subroutine for the player. However, we will still need to create a new player method to remove stuff from the inventory.

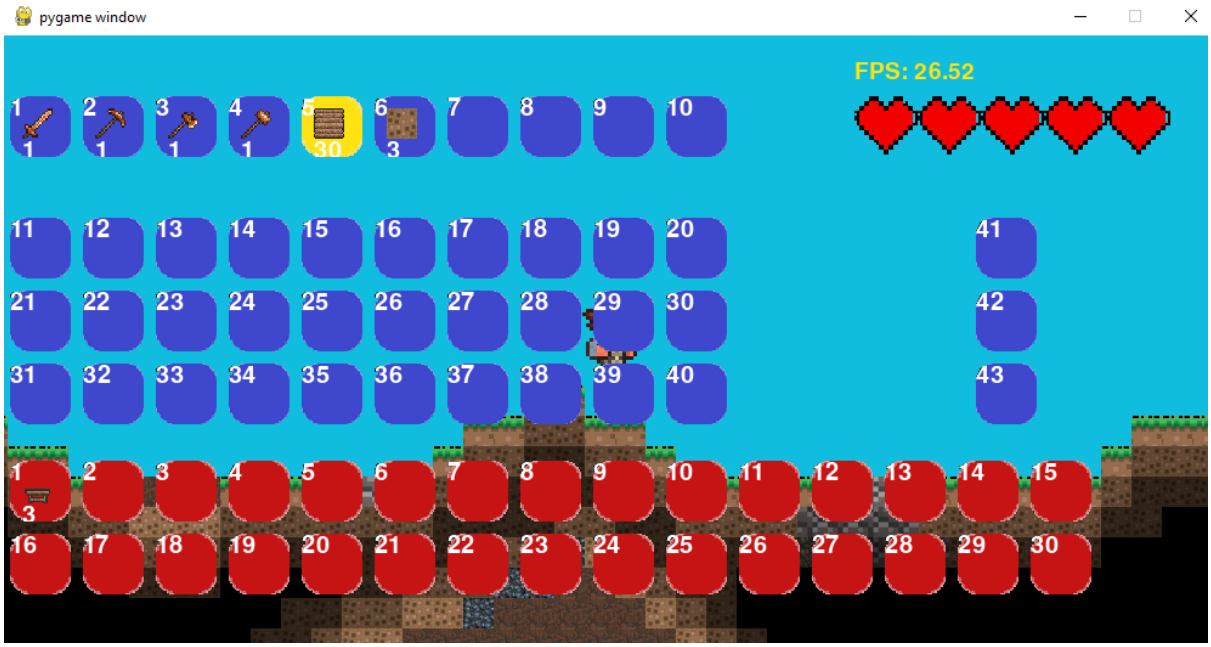
Logic for the addition to hud events-

If the class name of the sprite = “craft unit” then hudclicked is True, if the itemName of the sprite clicked on is “empty” do nothing, otherwise pass that item name into the player.craft method.

Player.craft:

```
DEFINE FUNCTION craft(item):
    tile=inventory.tile
    ingNameList=list(inventory.recipeDict[tile][item].keys())
    ingCountList=[inventory.recipeDict[tile][item][ing] FOR ing IN ingNameList]

    FOR index, ing IN enumerate(ingNameList):
        FOR unit IN inventory:
            IF unit[1]==ing:
                ingCountList[index]-=unit[2]
                CALL invChange(unit[0], unit[1], 0)



        CALL addToInv(ingNameList[index], -ingCountList[index]))

    addToInv(item,1)
    inventory.updateCrafting()
```

After an item is crafted or esc is pressed again the menu will be refreshed and crafting menu recalculated.




## Objective 11) Enemies:

This is another integral part of the game as enemies provide challenge which is a key factor for immersion into the game. This is because enemies will constantly attack and so you must keep your guard up. Meaning that there is no opportunity to get bored.

### Outline:

There will be three enemies in the game, the slime which comes out in the daytime and tends to stick in the same location, whereas the zombie and skeleton which are much tougher and will come out in the night and will travel in one direction until they find you.

### Activity:

Slime comes out at daytime

Skeleton and zombie spawn at night-time

Behaviour while not seeing player:

Slimes tend to stick in one area while occasionally jumping

Skeleton and zombies will roam the map in one direction and then flip to the other side if they hit the edge

Behaviour while seeing player:

Slimes will aggressively jump and chase player, jumps being enhanced, and speed increased.

Zombies will run toward the player faster

Skeletons will run toward the player occasionally shotting a high-speed fireball

How they deal damage:

All enemies deal damage on collision with the player of various levels

Skeleton creates a fireball enemy object which then collides and annihilates to deal damage.

## AI structured English-

### Mutual (To slime/zombie/skeleton)-

Always move forward in direction facing

Will not sink into the floor or walls by utilising identical collision as the player

If within x meters of player then attack

If attack is True:

Face player

speed increase

Jump increase

If collide with player, deal damage, and then set a timer for attack cooldown

While attack cooldown is active, colliding with the player will not cause damage

Decrement any timers per update loop

Every enemy will take knockback on damage as a velocity which is a% of the damage dealt over the remaining hp of enemy, enemy is also stunned for a short duration setting a cooldown timer off which when gets to zero allows enemy to move.

Therefore, as the enemy gets weaker, they will be knocked further back.

When enemies take damage the % transparency of the image will depict the remaining health of the enemy

Enemies do not take fall damage.

## Slime-

30hp

Spawn at day

Always play the first few frames of the jumping animation to make the slime appear as if shaking like jelly.

If collide with wall, then turn the other way

Jump randomly when cooldown is zero (height depending on if attack=True or not);

Set a timer for jump.

On jump play jump animation and stop the animation when it finishes

If attack is True:

If collide with wall, then jump instead of changing direction

Jump timer decrements faster

## Zombie-

100hp

Always play the walking animation

If collide with wall, then jump over it

Have a small chance to drop uru and adamantite on death

## *Skeleton-*

200hp

Play the walking animation when not shooting

If collide with wall, then jump over it

If attack is True:

Shoot player with fireball at an angle based on player and skeleton position

This angle is calculated by using pygame.math.vector2 data structure and then using

skeletonPos.angleTo(playerPos) which gives angle between skeleton and playerPos in radians

While shooting, the animation switches to the skeleton using animation and it stands still while casting. Once finished casting the fireball will be shot and he will start running again. Set a timer for attack cooldown. Have a small chance to drop uru and adamantite on death

### Fireball-

Move forward at constant velocity at a certain angle from the caster

for certain velocity V of projectile separate it into components:

V=some value

vel.x=Vcos(angle)

vel.y=Vsin(angle)

Will move forward until 100 frames has passed or the ball collides with something

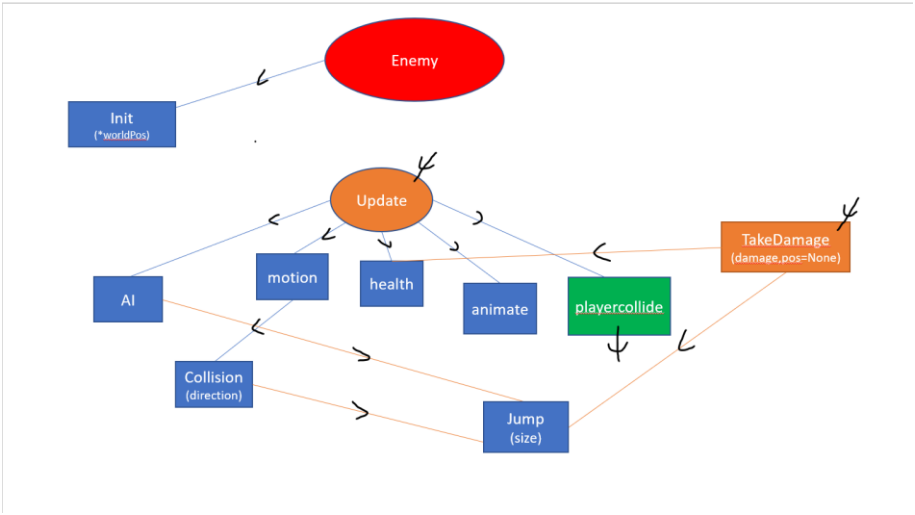
If the ball collides with a personSprite then it will deal damage to that sprite before annihilating

Ball is animated in a way that it looks like it is sparkling

Spritesheets

## Enemy Class diagrams-

For slime and zombie:




Red is the colour of object class

Blue means method is called from inside class

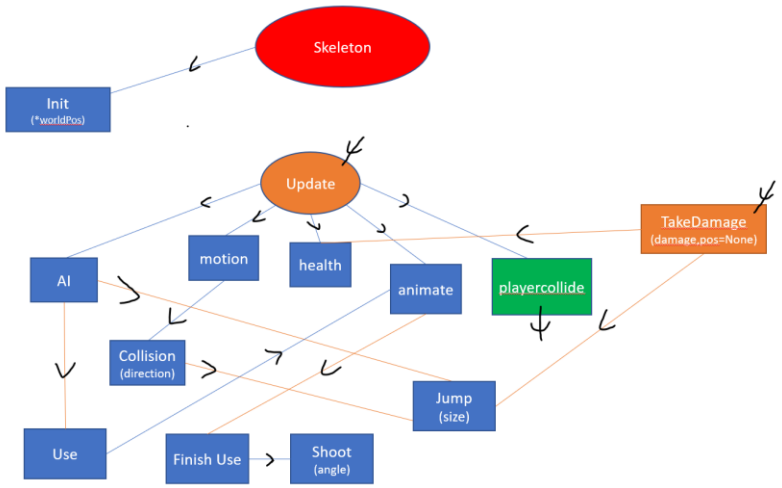
Orange means method is called from outside the class

Green means method calls something outside of class

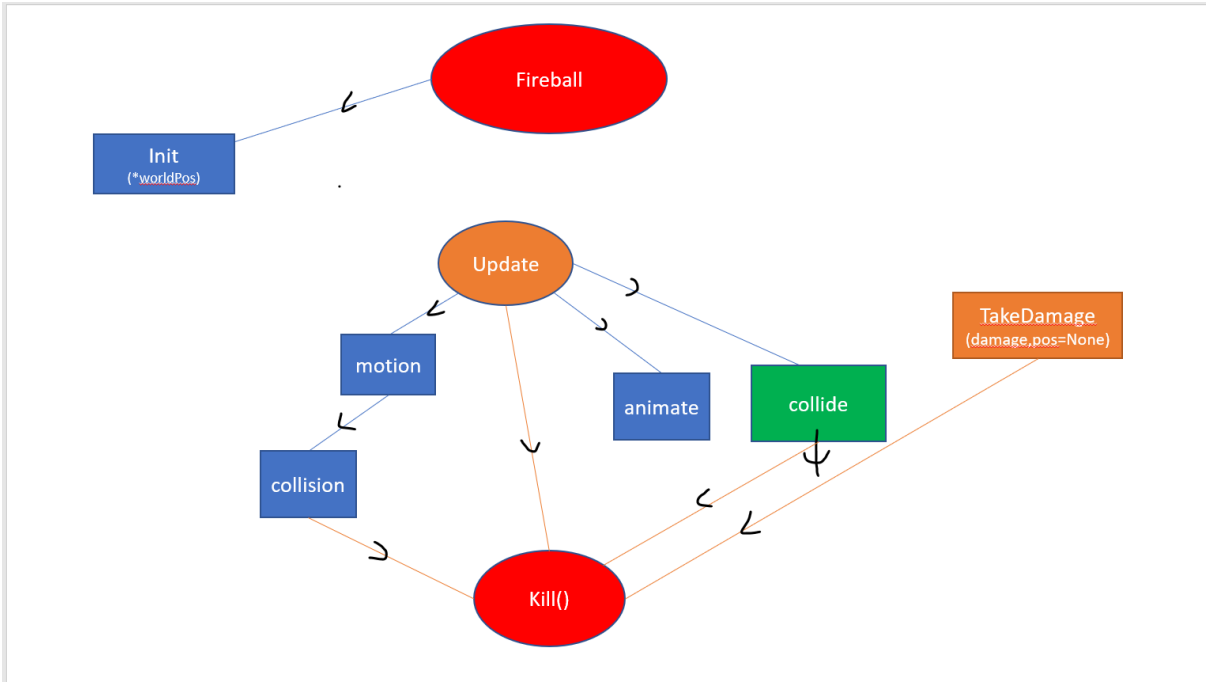
Blue lines with arrows mean method is always called and arrows show from where.

Orange lines with arrows means that method is sometimes called

For skeleton class




For Fireball




Here I have included the kill method as well to show how projectile can annihilate

### Dealing damage to enemies

If a hit takes place, then the enemy should take damage and be knock backed in a certain direction

For this we can use the checkCollision method for the tool class that we have already modelled.

However, an extra argument will need to be passed into takeDamage method of the enemy as the enemy would not know in which direction the damage was dealt and so would not know in which direction to be knocked backward. This means we need a new positional argument “left” or “right” depending on which direction the player is facing. This positional argument tells the enemy which way it needs to be knocked back.

Furthermore, the knockback needs to be of a certain velocity which has not been specified by the tool class. However, the knockback to be realistic needs to be proportional to the fraction of damage dealt by an attack and the hp an enemy has left. This is because a stronger enemy should be more resistant against an attack than a weaker enemy and as the enemy gets weaker it should take more knockback.

## Objective 14) Animations

Adding animations is a major part of increasing the amount of immersion in a game, which is one my most important features for this game.

Animation is simply switching between images quickly enough for our human eyes to perceive something moving. This is normally done in games by using a sprite sheet and then exporting the sprite sheet to a list of images but iterating across each frame.

Loading Sprite sheets

https://www.youtube.com/watch?v=M6e3_8LHc7A

I watched this video to understand how to load sprite sheets

Sprite sheet Logic-

Open the file

For each frame in the sprite sheet iterate across the width of sprite sheet using the width and height of each frame. We can use the area argument of the pygame.surface.blit fuction as it allows us to pick a starting position and then an area to select for printing onto a surface. This image can then be printed onto a surface of size width and height, giving us a frame in the sprite sheet.

A colour key is needed to make the region of transparency around the frame become transparent in python.

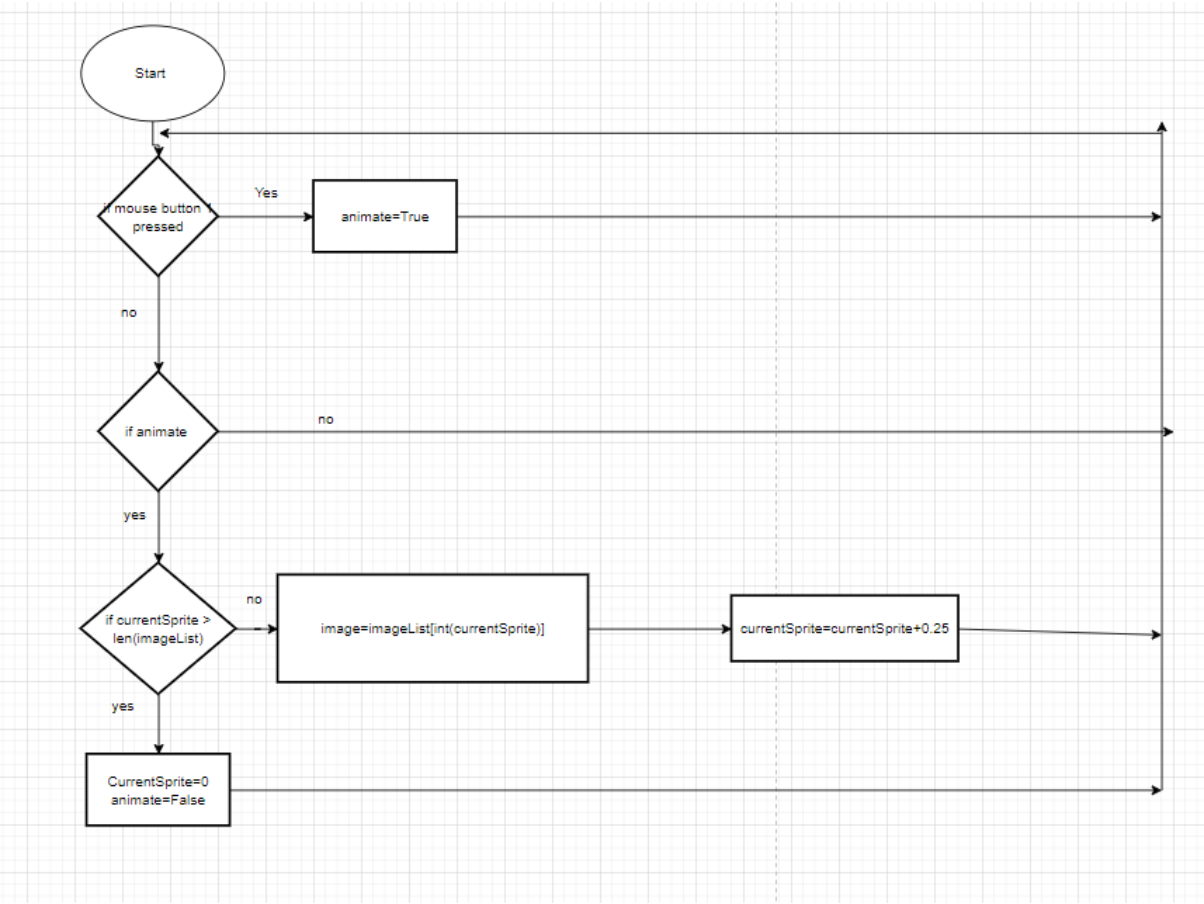
A scale can also be applied If the images on a sprite sheet are too small to be used in your game.

Sprite sheet python code:

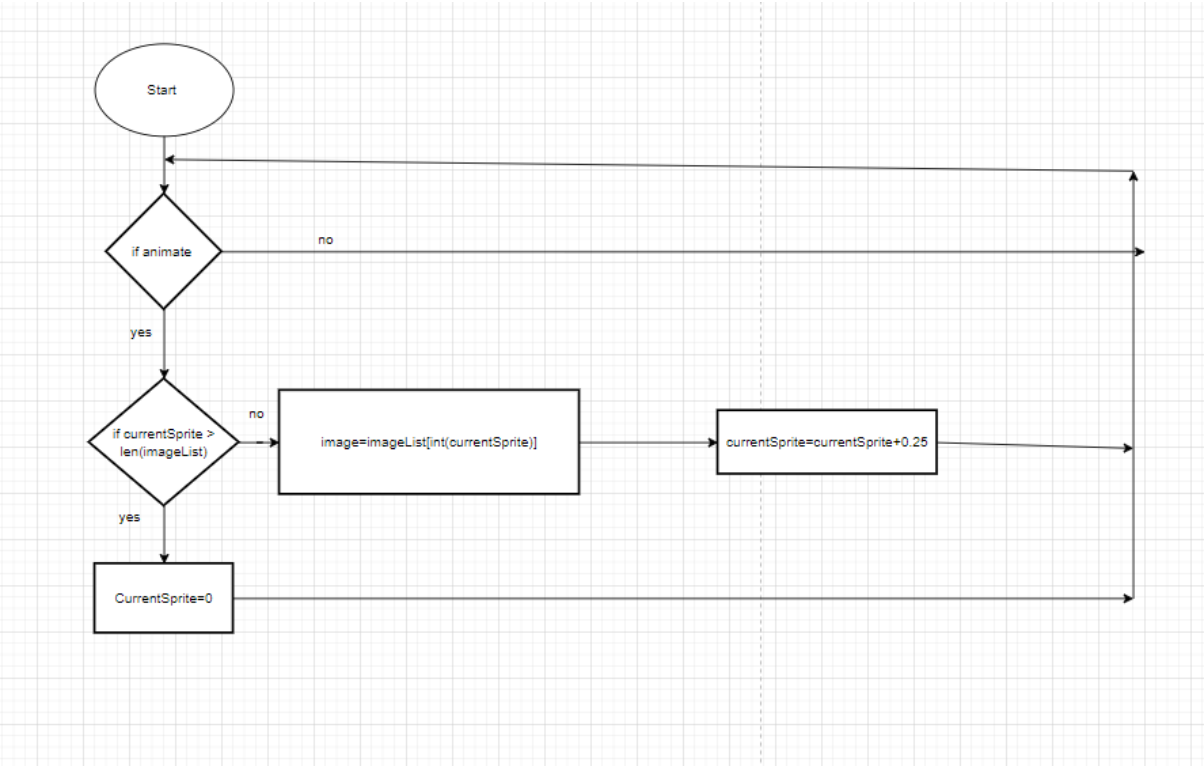
```
class Spritesheet:
    def __init__(self, filename, scale, colour, width, height):
        self.filename = filename
        self.spriteSheet = pygame.image.load(os.path.expanduser(filepath + filename)).convert()
        self.scale=scale
        self.colour=colour
        self.width=width
        self.height=height

    def getImage(self, frame):
        image = pygame.Surface((self.width, self.height)).convert_alpha()
        image.blit(self.spriteSheet,(0,0),((frame*self.width), 0, self.width, self.height))
        image = pygame.transform.scale(image, (self.width * self.scale, self.height * self.scale))
        image.set_colorkey(self.colour)

        return(image)
```

Once we have the list animation is simple and just comes down to.

Flow chart for an on click animation:




Flow chart for a constant animation:



Keeping in mind that animate will be a public method of an object which can be called on certain events. For example, If the player walks left then animate for walking left, else animate for walking right, or if the player jumps then override the current animation with the jumping animation.

# Testing

## Objective 1 Menu

Menu testing video:

Menu Test.mp4

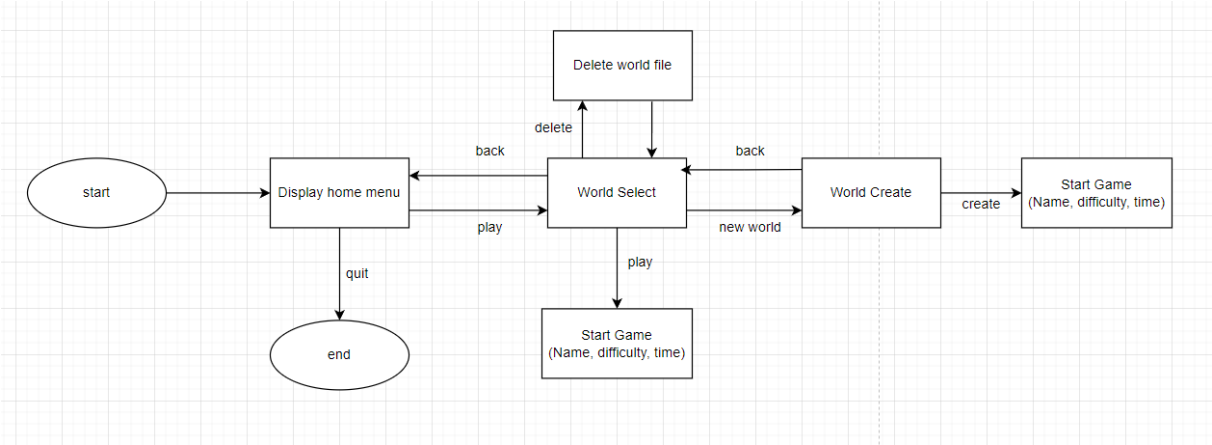
1) First, I am testing the buttons and whether they highlight if you hover over them.

   Expected result:
   If the mouse cursor is over the rect of button it highlights, once the mouse leaves the rect of the button it stops being highlighted.


Valid Test: If your mouse cursor goes for the button is highlights

Result: view vid 1
Valid Test is successful

2) Pressing the buttons will take you the correct menu screen:
Expected result:




Result: view vid 1
Valid test=click on a button → take you to correct screen

View menu test video

Result: test is successful

3) Hovering over an entry will cause it to highlight:
Expected Result: will cause entry to fill yellow


Result: test is successful

4) Left Clicking on a world object:
Expected Result: will cause it to fill red and display new buttons play and delete

Valid Test: click on the world object→ expected result



5) Left Clicking on a nameEntry object:
Expected Result: will fill red and allow you to type into the box

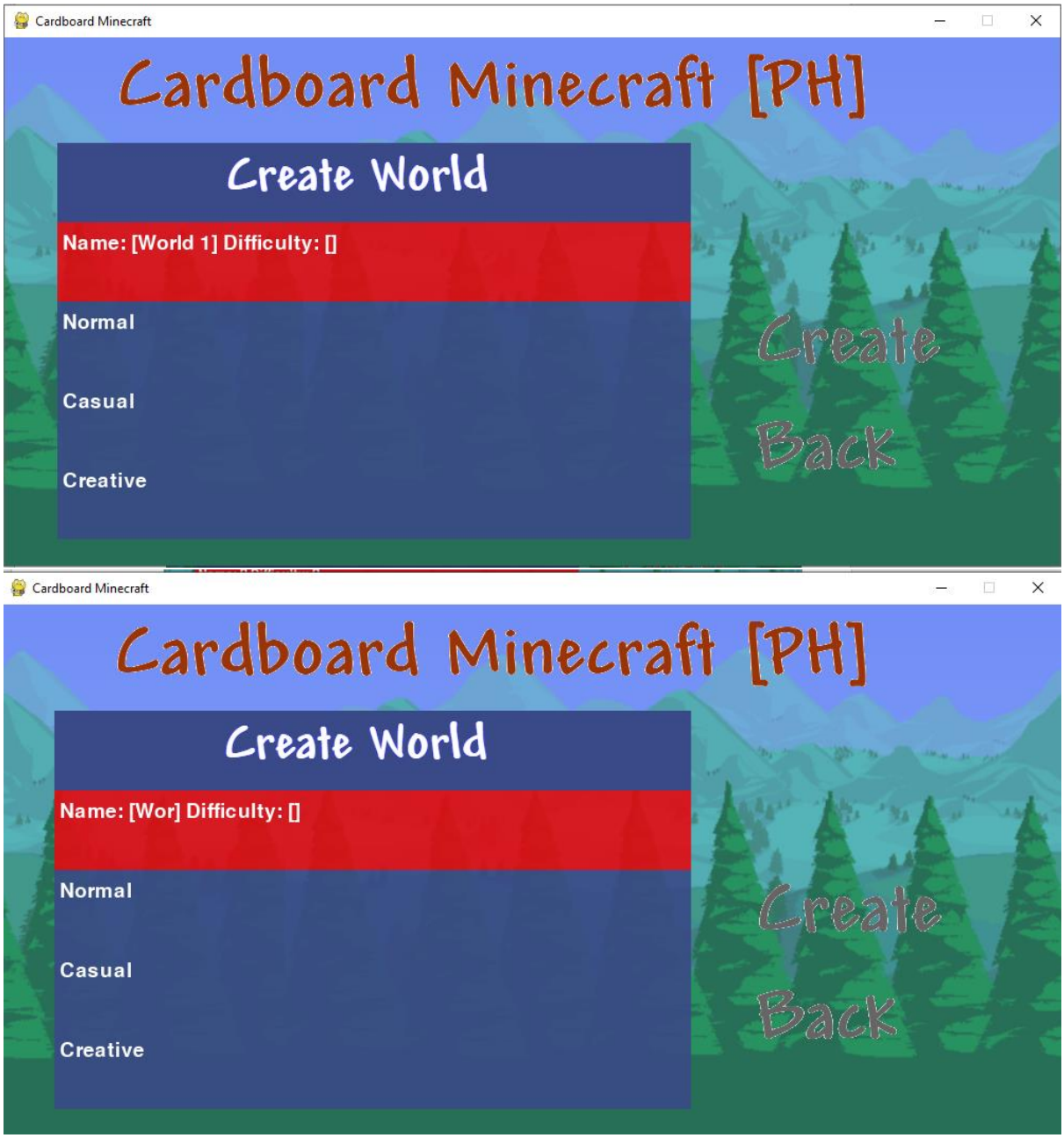
Valid test: Click on the nameEntry object and try to type into the box, test backspacing as well.

Result: test is successful- view menu video


6) Does loading and saving files work?
Expected Result: When I load a world that I have played before my inventory, position, health, and the position of all objects is conserved

saving and loading world.mp4


FPS: 30.21
FPS: 30.12
Cardboard Minecraft - Microsoft Game DVR




6) Left Clicking on a difEntry object:
Expected Result: will fill red and the difficulty will appear inside the nameEntry box
Valid Test: Left Click



Result: test successful

7) What happens if you click on create world without selecting anything
Expected result: Clicking create world without selecting anything will put you into a blank world named “world” difficulty: “normal”

Valid test: click on create world


Result: Test failed, world is created with no world argument and no difficulty.

8) What happens if you try and more than 4 worlds
Expected Result: error message displayed

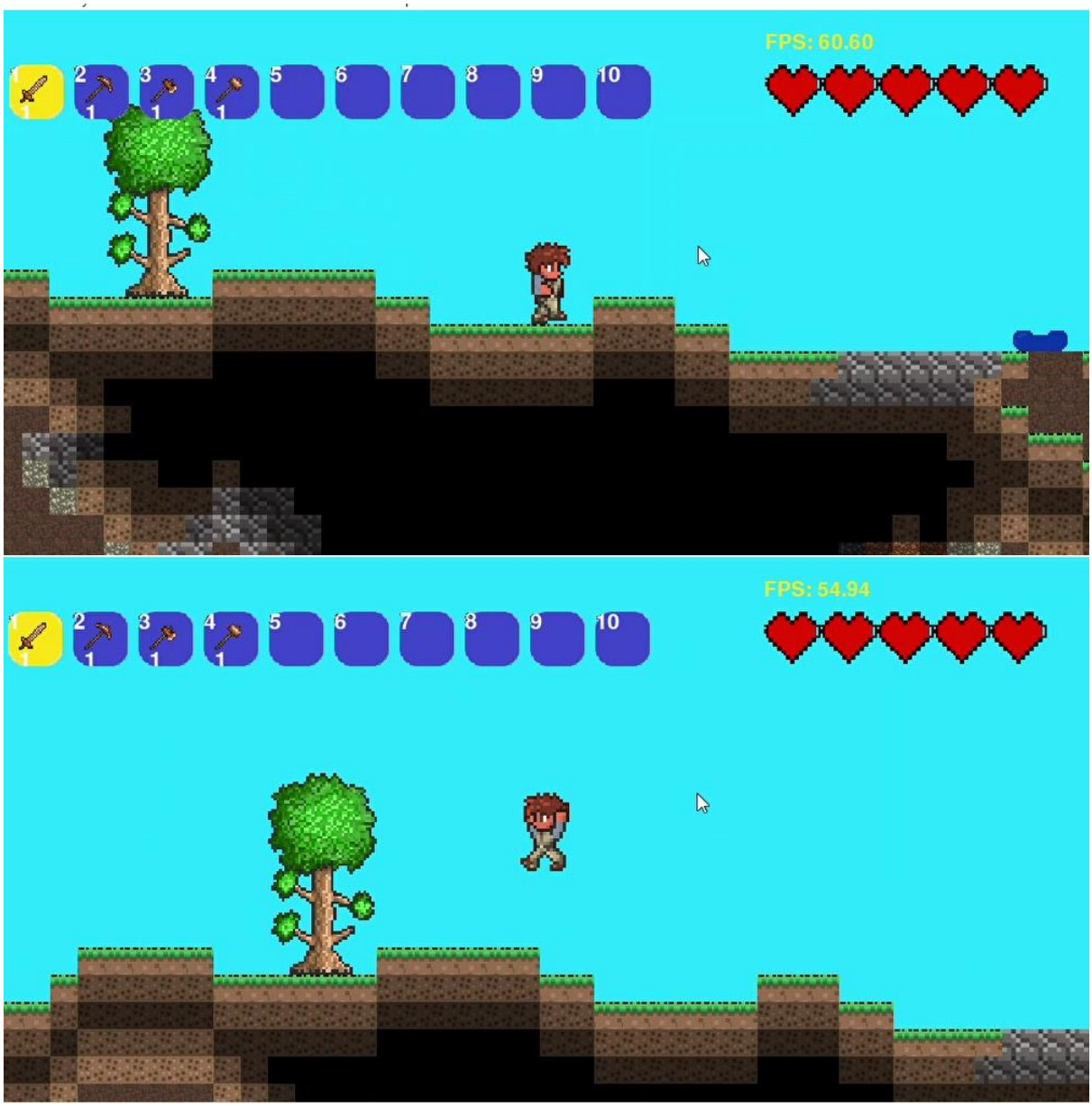

Result: test successful

## Objective 2 Player motion and Camera

Player Motion Test and Camera Test:

Cardboard Minecraft 2022-04-1

1) Does player move right and left and jump when you press “a”, “d” and space:




Result: test is successful

## Objective 3 collision:

Cardboard
Minecraft 2022-04-1

1) Does the player collide with the floor, walls and ceiling?
   Expected result: player collides with floor, walls and ceiling


Result: test is successful

## Objective 4 Tiles:

Tiles test.mp4

1) Can blocks be placed and destroyed?
   Test: Destroy a dirt block and place the dirt block

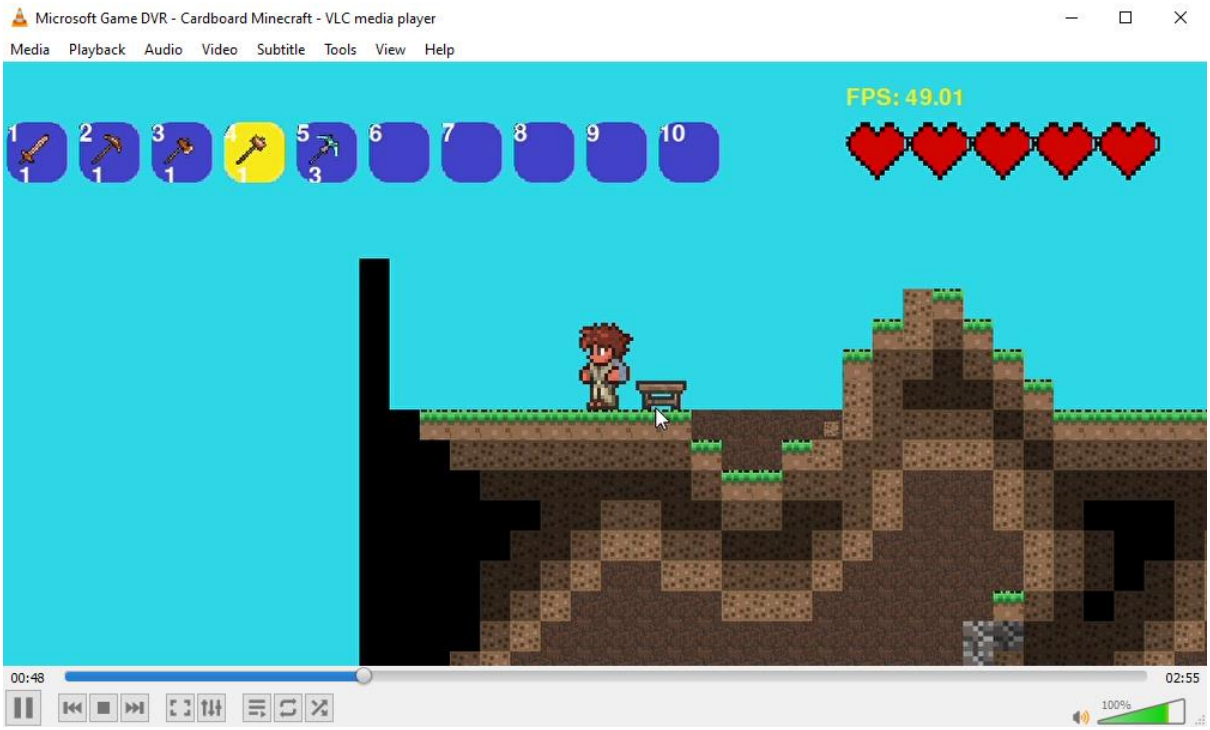
   Expected result – block is destroyed by pickaxe only and can be placed when held in hand.

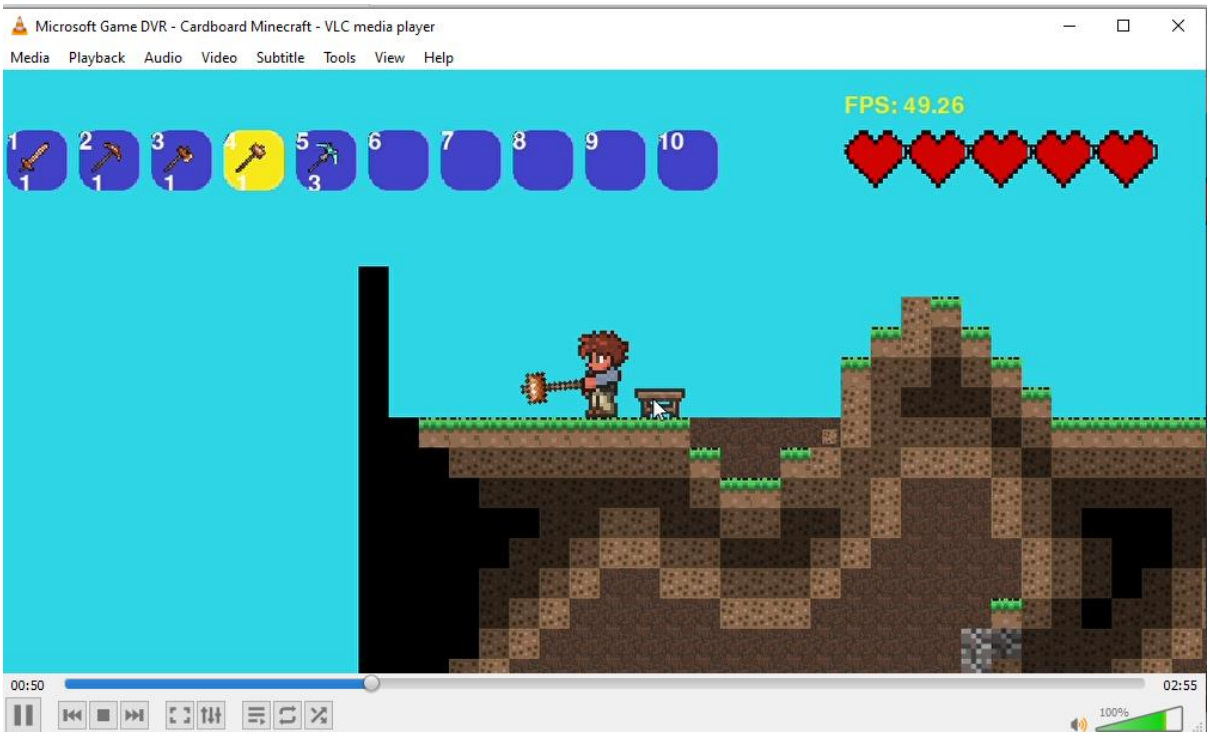

Test is successful

2) Can trees be destroyed?
   Test destroy a tree
   Expected result: tree can be destroyed be axe only


Result: Test is successful

3) Can tiles be placed and destroyed?
   Test: Place a workbench and destroy it
   Expected result: workbench can be placed if in hand and destroyed by hammer only.






Result: test is successful

4) Can walls be placed and destroyed?

Test: place a wall and destroy it

Expected result: wall can be placed if in hand and destroyed by hammer only.


Result: test is successful

5) Do blocks have different health values – some blocks take longer to destroy?
   Test: break a dirt block and a tin block see whether the tin take longer than the other.
   Expected result tin takes more hits to destroy than dirt and both can only be destroyed by pickaxe

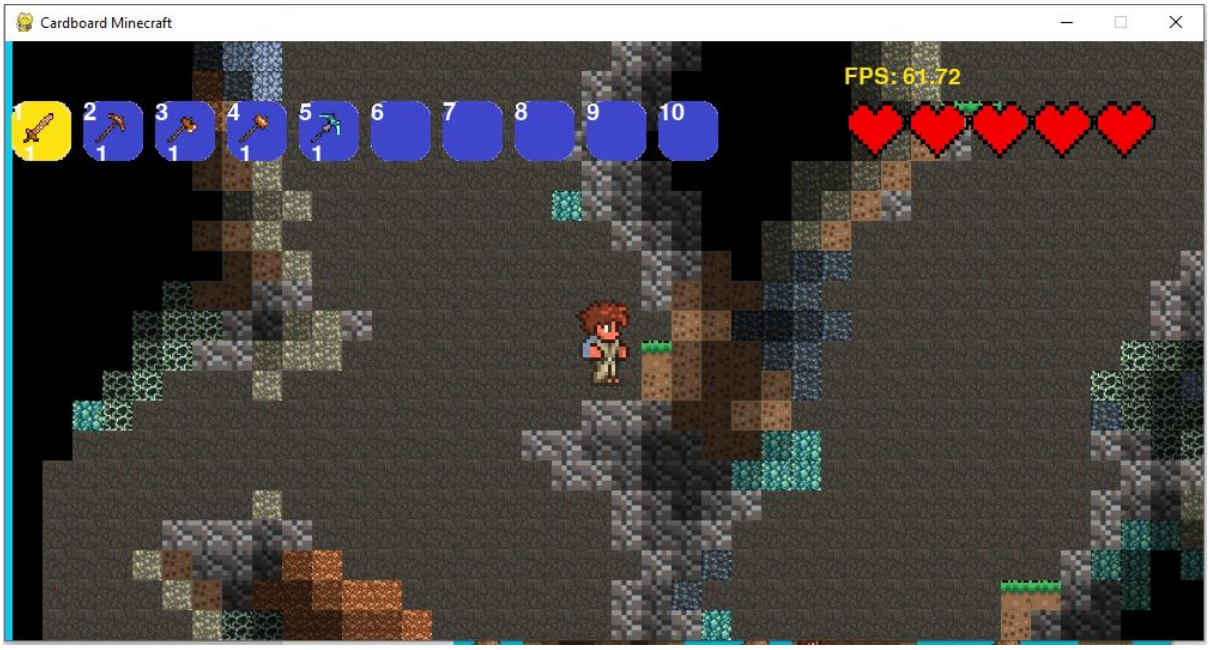

Result: test successful

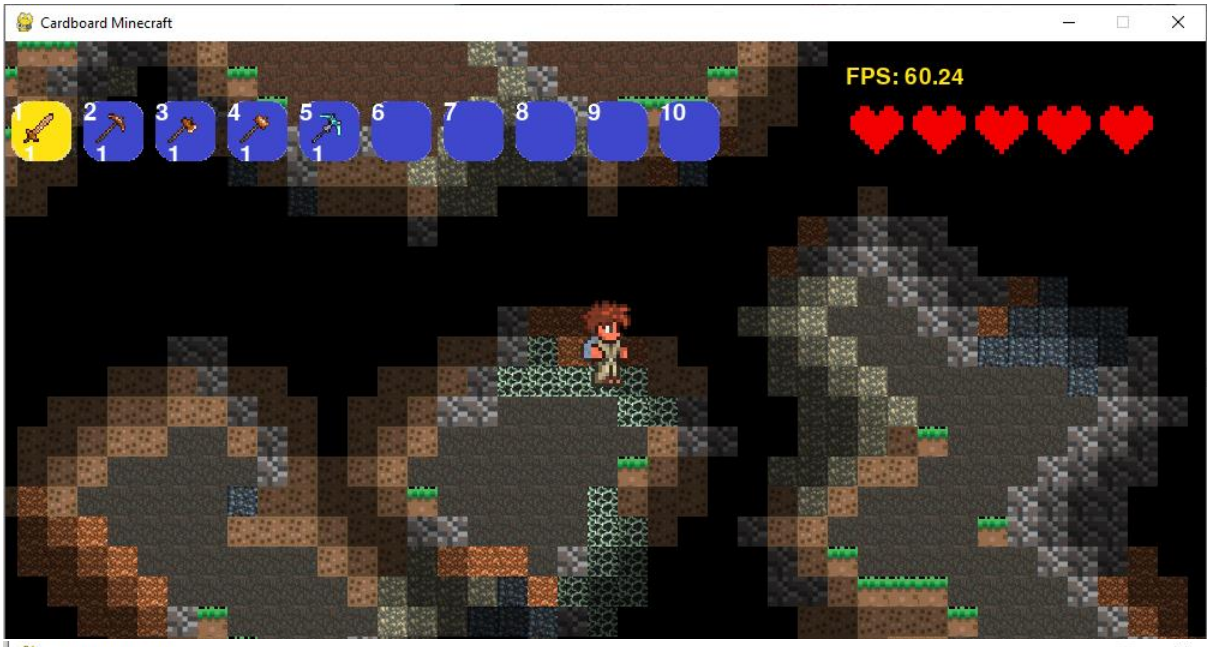
## Objective 6 Procedural Generation:

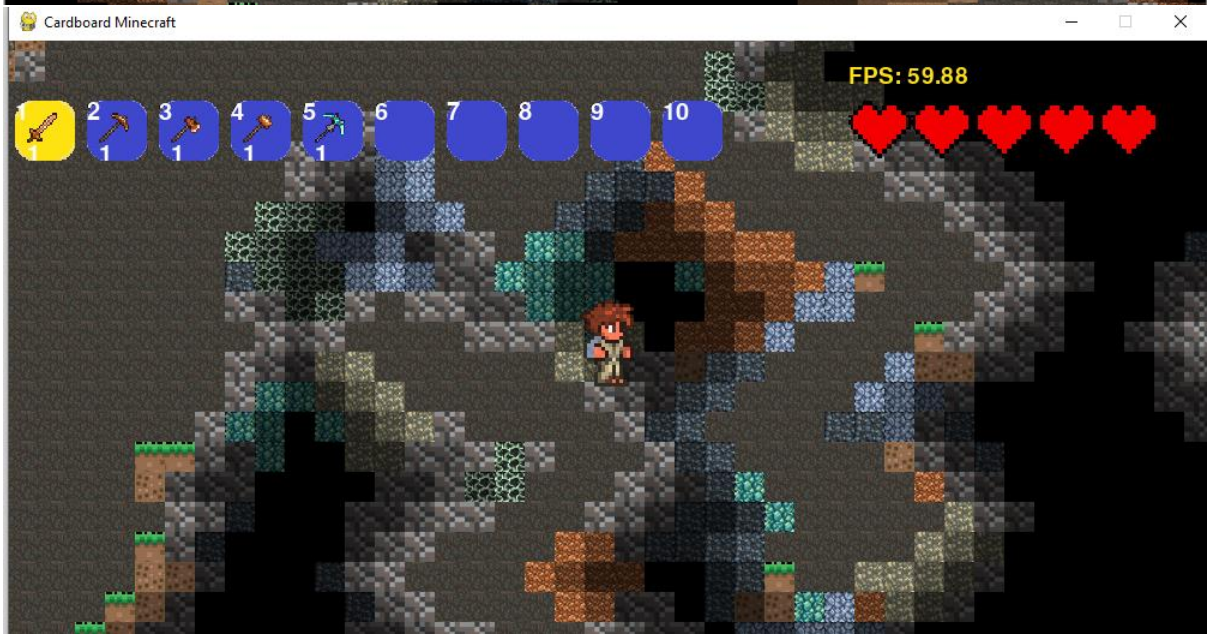
Procedual generation Test.mp4

1) Does the procedural generation create the world correctly?
   Expected result:

   - Different patches of stone and dirt and generated.
   - Small patches of Ore are generated
   - Caves generated
   - Trees generated
   - Darkness working


Cardboard Minecraft
FPS: 61.72
1
2
3
4
5
6
7
8
9
10



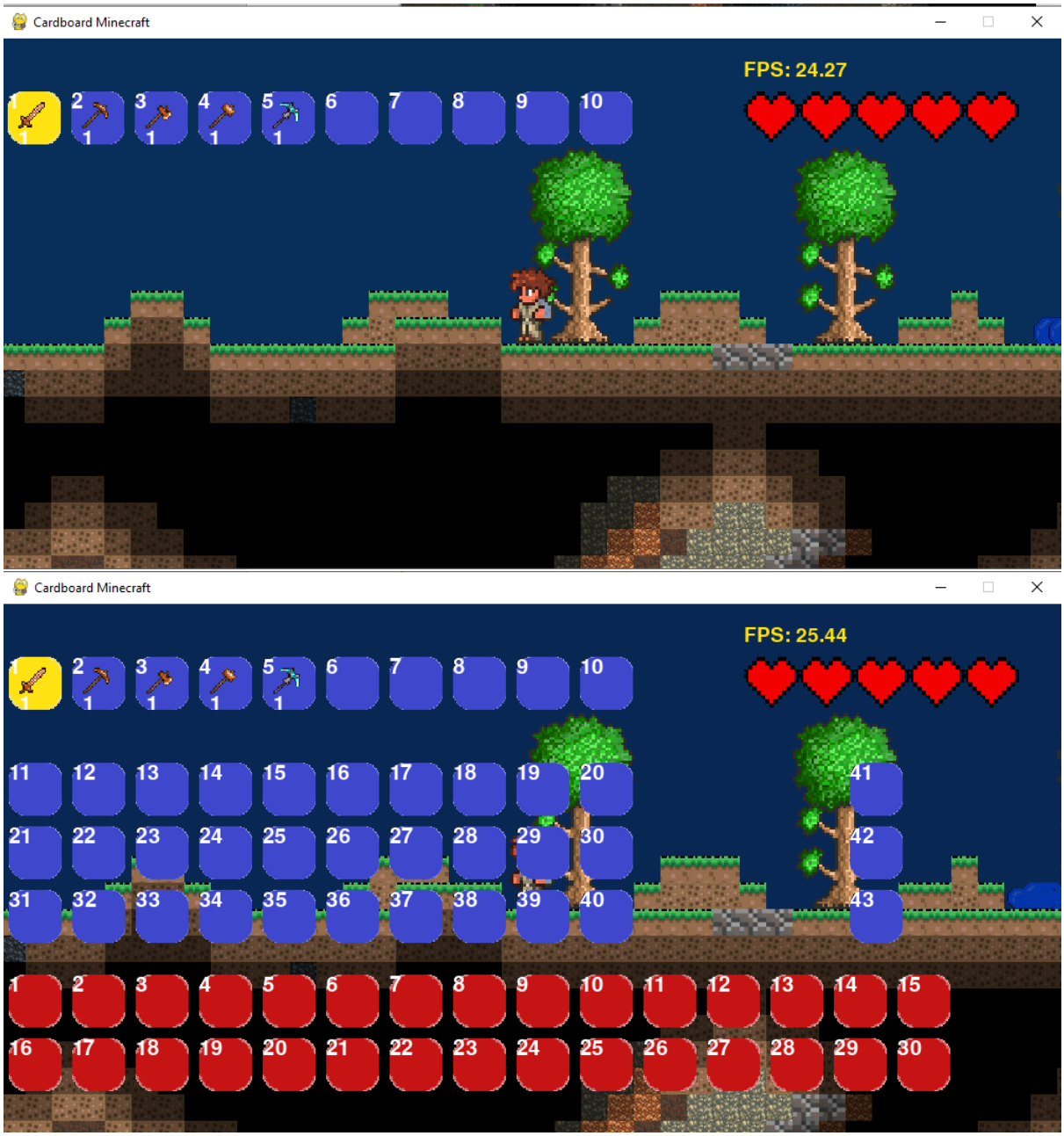




Result: test is successful

## Objective 7 HUD

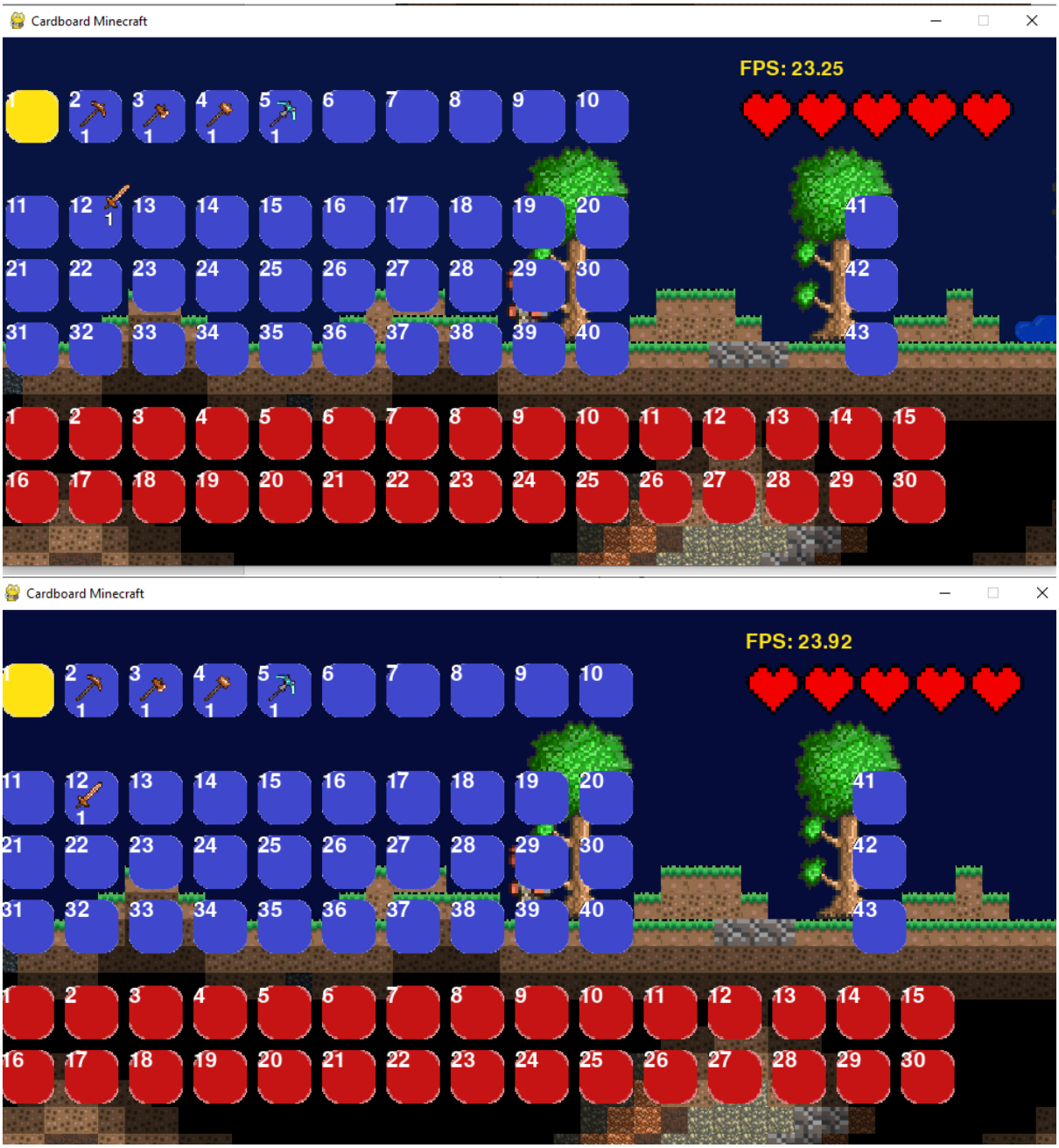
Inventory Test.mp4 Health Bar test.mp4

1) Does inventory appear on ESC pressed
   Expected result: on ESC press the inventory is appear




Result: test successful

2) Can you move an item around in the inventory?
Expected result: Clicking on an item will add it to your hand and you can place it in empty slots in the inventory or you can swap what is in your hand with something, if you use right click then it will pick up and drop a single item.




Result: test successful

3) Can you drop items from the inventory?
Expected result: Using right click outside of the inventory while having an item in hand drops the item at that position


Result: test successful

4) What happens if you have an item in hand and use left click outside of the inventory
Expected result: item returns to the place it was picked up from


Result: test successful

5) Does the health bar go down if the player takes damage?
Expected result: the health bar will go down from fall damage and taking damage from enemies.

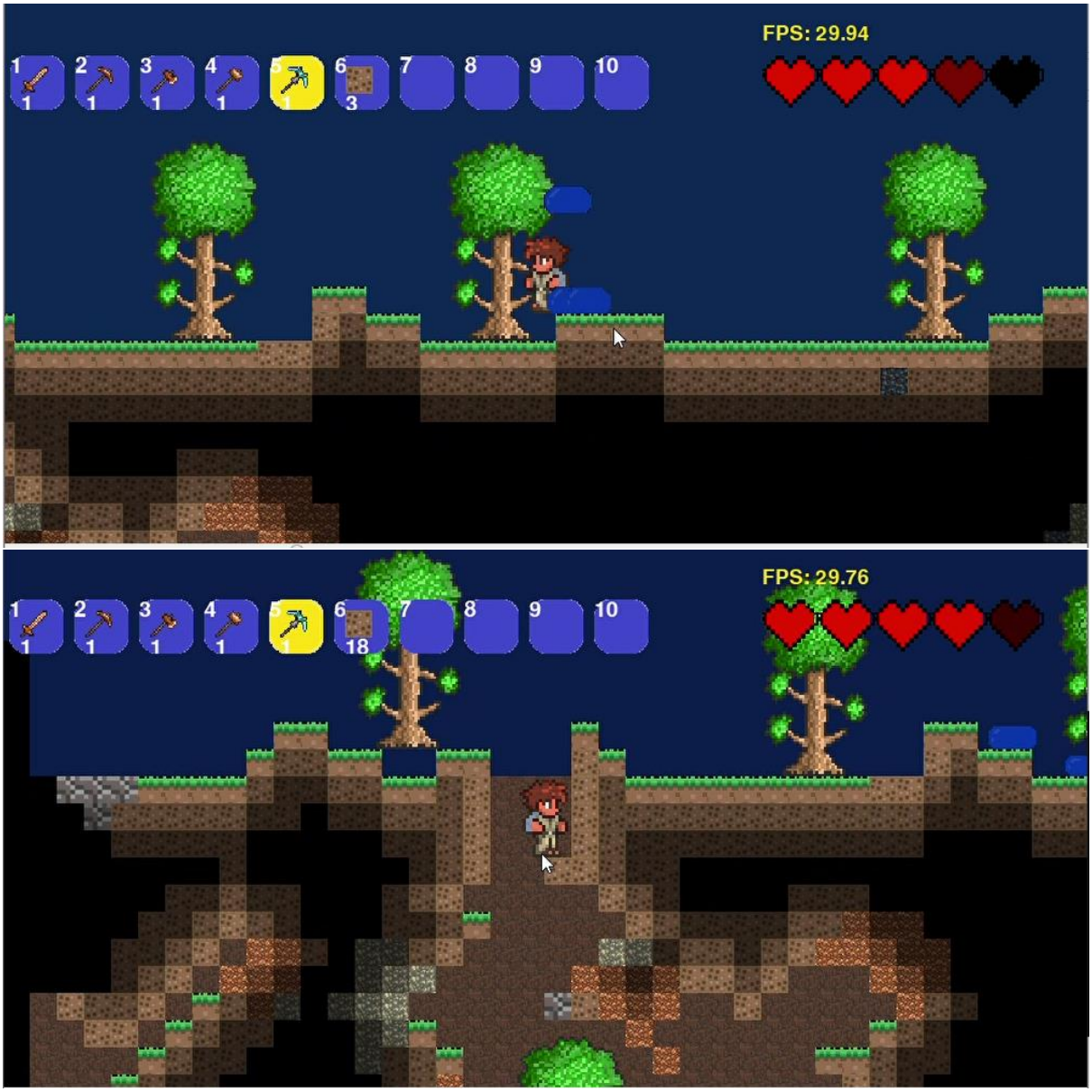



Result: test is successful

6) Does the amount of health display if you hover over the health bar?
Expected result: the exact health/maxhealth will be displayed if you hover over the health bar.


Result: Test failed, the message is not shown

7) Does the health bar regen after not taking damage?
Expected result: after waiting a bit after taking damage the health should start to regen.




Result: test is successful

## Objective 8 Player inventory and health

Testing for is section would be identical as the HUD testing

## Objective 9 Items

item test.mp4

1) Do the tools swing properly?
   Expected Result: Swing animation is carried out properly


Result: test successful

2) Do the tools break their corresponding tiles? - view objective 4 test

3) Do tool swings do damage to enemies?

   Expected Result: any tool will do damage to enemies if the enemy mask collides with the mask of the tool


Result: test successful

4) Do weapon projectiles work?
   Expected result: projectiles will shoot from the tool as long as it is a projectile shooting weapon and only once per tool object life time.


Result: test successful

## Objective 10 Crafting

crafting normal test.mp4

1) Does the crafting menu appear after pressing ESC?

Expected Result: crafting menu will refresh every time esc is pressed

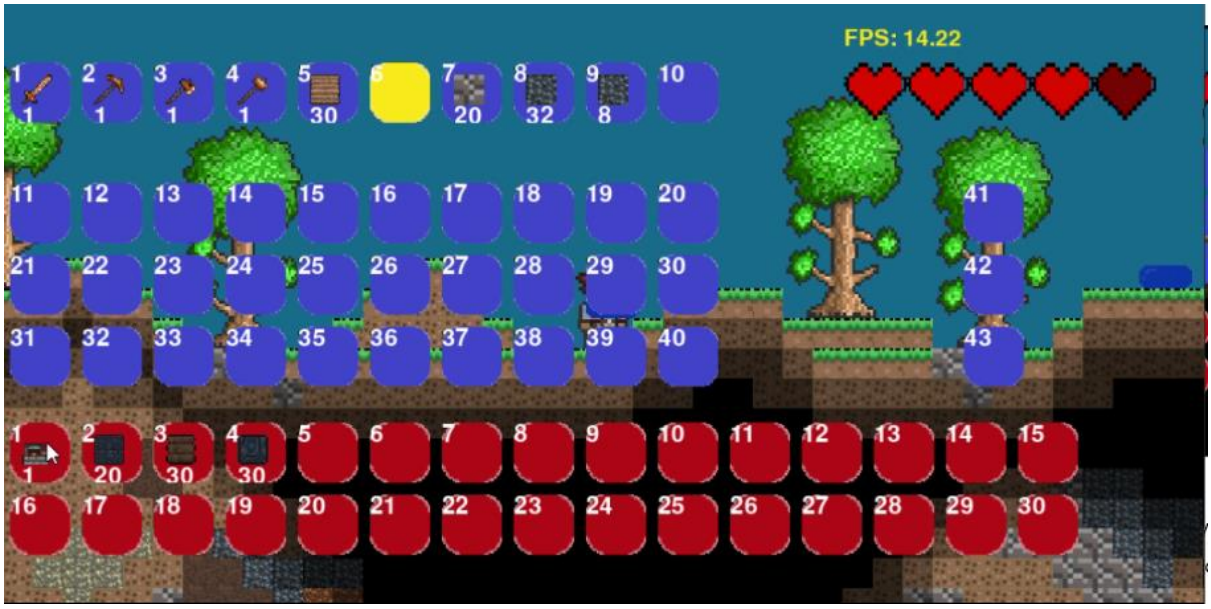

Result: Test Successful

2) Do the relevant recipes appear if the player has the correct materials and they are standing next to the correct tile

Expected Result: relevant recipes will appear if that play is colliding with the correct tile and has the correct amount of materials


3) Do the recipes clearly show the resulting material and how many of them the player can still craft

Expected Result: Recipes will show how many of the resulting material can be crafted based on the limiting ingredient in the crafting.




3) Can the player obtain the resulting crafted item and lose the ingredients from their inventory.

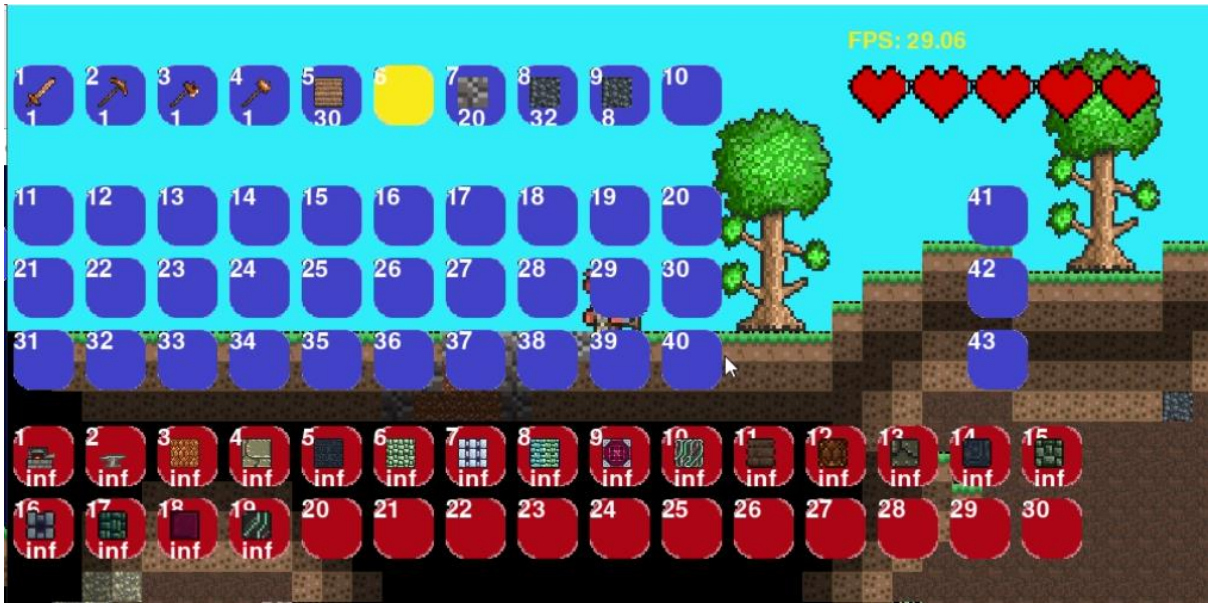
Expected Result: In a transaction the player will first lose the materials needed to make that recipe but will in turn gain the item that they chose to craft and the excess materials that were not used in the recipe.


Result: player lost 20 stone in crafting the furnace, test is successful

crafting creative test.mp4

4) is the player able to see all crafting recipes without the required materials in creative?

Expected Result: if the player is still colliding with the correct tile all the recipes should show without having the necessary materials. Also, they should show infinity for the number of items craft able.




Result: Test is successful

## Objective 11 Enemies

slime enemy test.mp4

night enemies .mp4

1) Does the slime perform its designed behaviour?

Expected result: slimes come out during the day and will stay in one area when not attacking the player but when the player is in their attack range they will aggressively jump and attack the player.


Result: Test successful

2) Does the Zombie perform its designed behaviour?

Expected result: zombies come out during the night and will walk in one direction either finding the player or despawning, once the zombies “see” the player they will turn toward the player and run faster and jump higher.


Result: Test is successful

3) Does the Skeleton perform its designed behaviour?

Expected result: skeletons come out during the night and will walk in one direction either finding the player or despawning, once the skeletons “see” the player they will turn toward the player and run faster and jump higher. They will also fire projectiles every few seconds.


Result test successful

## Objective 12 Progression

1) is there an appropriate progression in the tool/weapon strength of different items?

- Pickaxes – Expected Result – stronger pickaxes will be able to mine stronger blocks and faster
- Swords – Expected Result – stronger swords will do more damage, faster attack speed and will fire projectiles
- 
- Axes – Expected Result – will take less hits to destroy tree and attack speed is faster
- Hammers – Expected Result – will take less hits to destroy walls/tiles and attack speed is faster

sword progression.mp4 pickaxe progression.mp4 axe progression.mp4 hammer progression.mp4

Result: Test successful


## Objective 13 events

1) Is there day and night with the relevant enemies?

Expected Result: day and night



Result: test successful

## Objective 14 animations

1) Do animations work correctly?

Watching the other videos should suffice

Test successful

## Objective 15 Difficulty

1) On normal is there an increased enemy spawn then casual?

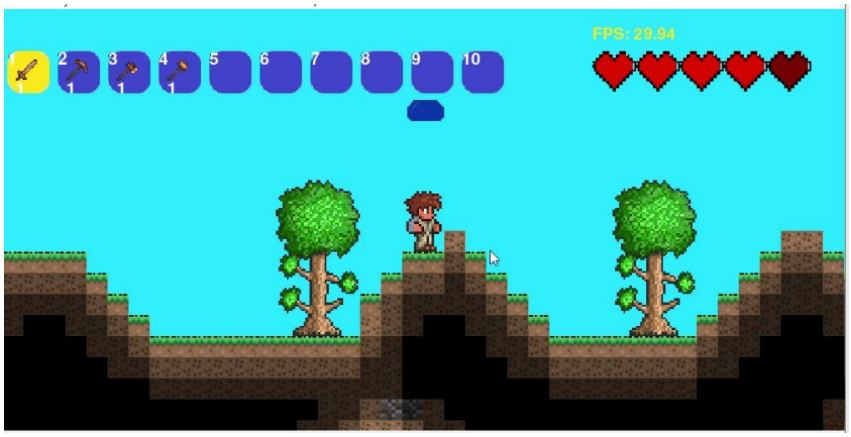
Normal enemy spawn and damage

normal mode.mp4


2) In casual is the player damage reduction and player regeneration higher than normal?

Casual enemy spawn and damage

casual mode.mp4




3) In creative can you fly/noclip/godmode/craft without requiring materials?

creative mode.mp4


Result: tests successful

### Objective 16 Optimisation

In the previous videos (before crafting) I was on a good computer and so even while doing videos I was getting 30 FPS capped.  View previous videos for reference. On the objective I mentioned that 30 is good enough and so I will call this test a success.

## Evaluation

Overall, I think that this project was a success as most of the objectives have been met.

The only objectives which have not been met would be the implementation of armour into the game, this is something I could design and implement. However, this wasn't possible under the time constraints in the NEA.

1) Menus

I think the menu was done to a great standard; the only thing that could have been improved upon is increasing the limit for the number of worlds from 4 to infinite but that would require a scrolling dropdown box. Furthermore, it is possible to impletement a settings menu but it practically useless on a project of this scale.

2) Player motion and Camera was perfect although there was possible some performance problems by utilising a camera group

3) Player collision was the best it could have been with pygame as the engine is not advanced enough to give perfect motion and collision. Sometimes it is possible to jump through wall and ceilings if the player is trapped. The only work around would be to use another engine

4) Tiles was done perfectly as well; I can't think of a way I can improve it other than making some tiles obey gravity or other rules such as they must be glued to the ground.

5) Materials there is not much substance to talk about as it is simply using some things as ingredients; therefore I did it perfectly.

6) Procedural generation was done to a good standard, the only issue would be the problem with performance in loading an entire chunk the size that I did, and so a smarter algorithm with chunks would be needed to create a large and optimised world.

7) HUD was ok, but I left the armour section without finishing it, furthermore the descriptions when you hover over an item or if you hover over the health bar wasn't there.

8) Player inventory and health was also done ok but I did not use protected attributes for them and so there are quite a lot of holes in it.

9) Items was done well apart from the fact I never finished the armour section because of time restraints.

10) Crafting was done to a solid standard as it was lightning fast and always worked.

11) Enemies were done to a solid standard as well as they acted intelligent (to an extent you would expect in a game such as terraria) and did a various number of things. This meant that they were a true challenge for a player playing on normal difficulty.

12) Progression was very good as there is a gradual increase in power for the player as they get better tools.

13) Events were also very good as day and night was perfect and it also included the different enemy spawns as well

Feedback-

*There is no incentive to get better tools as there is no big boss to the game* - classmate

## Code

```
1. import pygame
2. import os
3. import sys
4. from time import time
5. import datetime
6. import random
7. from pygame.locals import *
8. from math import*
9. import json
```

```
10.     from extras import *
11.
12.     filepathG = os.getcwd()+"\\Textures\\gameSection\\"
13.     filepathM = os.getcwd()+"\\Textures\\Menu\\"
14.     filepathW = os.getcwd()+"\\Worlds\\"
15.     filepathJ = os.getcwd()+"\\Config\\"
16.
17.     TILESIZE = 25
18.     WIDTH = TILESIZE*40
19.     HEIGHT = TILESIZE*20
20.     WORLDLENGTH=3
21.     WORLDHEIGHT=3
22.
23.     pygame.init()
24.     FPS = 30
25.     fpsClock = pygame.time.Clock()
26.
27.     screen = pygame.display.set_mode([1000, 500])
28.     pygame.display.set_caption("Cardboard Minecraft")
29.
30.     #game fuction
31.     def game(screen, worldName="New World",
   difficulty="Normal", prevTime=0):
32.
33.         class Game:
34.             def __init__(self, screen, worldName, difficulty,
   prevTime):
35.                 pygame.init()
36.                 #initialise variables
37.                 self.clock = pygame.time.Clock()
38.                 self.screen = screen
39.                 self.worldName=worldName
40.                 self.difficulty=difficulty
41.                 self.time=self.startTime=prevTime
42.                 self.timeStopped=False
43.
44.                 self.noclip=True if self.difficulty ==
   "Creative" else False
45.
46.                 self.tick=0
47.                 self.timePeriod=60*10
48.                 self.colorFader=colourFader()
49.                 self.font =
   pygame.font.SysFont("AndyBold",28)
50.
51.                 #load spritesheets
52.                 self.load()
53.
```

```
54.                 #def sprite groups
55.                 self.hudSprites = HudGroup(self)
56.                 self.camSprites = CameraGroup(self)
57.                 self.personSprites = CameraGroup(self)
58.                 self.backSprites = CameraGroup(self)
59.                 self.backgroundSprites =
  pygame.sprite.Group()
60.                 self.colSprites = pygame.sprite.Group() #
  sprites that collide with the player
61.                 self.itemSprites = pygame.sprite.Group()
62.                 self.treeSprites = pygame.sprite.Group()
63.                 self.wallSprites = pygame.sprite.Group()
64.                 self.tileSprites = pygame.sprite.Group()
65.                 self.enemySprites = pygame.sprite.Group()
66.
67.                 #try to load the world
68.                 self.loadWorld()
69.
70.                 #load player
71.                 self.player = Player(self, *self.playerPos,
  self.playerInventory, *self.playerHealth)
72.
73.                 #darkness
74.                 for block in self.colSprites:
75.                     self.updateDarkness(block)
76.
77.                 #load healthbar
78.                 self.healthBar=HealthBar(self, 700, 50, 2,
  self.player.maxHealth, self.player.health)
79.
80.                 #load inventory objects
81.                 self.hand=None
82.                 self.inv=Inventory(self)
83.
84.                 #start time
85.                 self.FPS = FPS
86.                 self.prevTime = time()
87.                 self.initTime = time()
88.                 self.dt = 0
89.
90.             def load(self):
91.                 #loading the blocks
92.                 blockSheet=Spritesheet("blocks.png", 1,
  (0,0,0), 25, 25)
93.                 self.blockImageDict=dict()
94.                 self.blockStatDict=dict()
95.
```

```
96.                 with
   open(os.path.expanduser(f"{filepathJ}blocks.json")) as f:
97.                     data=json.load(f)
98.
99.                 self.blockStatDict=data
100.
101.                for index, block in
   enumerate(self.blockStatDict.keys()):
102.
   self.blockImageDict[block]=blockSheet.getImage(index, 0)
103.
104.                #loading the tools
105.                toolSheet=Spritesheet("tools.png", 1,
   (0,0,0), 50, 50)
106.                self.toolImageDict=dict()
107.                self.toolStatDict=dict()
108.
109.                with
   open(os.path.expanduser(f"{filepathJ}tools.json")) as f:
110.                    data=json.load(f)
111.
112.                self.toolStatDict=data
113.
114.                for index, tool in enumerate(data.keys()):
115.
   self.toolImageDict[tool]=toolSheet.getImage(index, 0)
116.
117.
118.
119.                #loading background
120.                backSheet=Spritesheet("background.png", 1,
   (0,0,0), 25, 25)
121.                backList=["dirt","stone"]
122.                self.backImageDict=dict()
123.
124.                for index, back in enumerate(backList):
125.
   self.backImageDict[back]=backSheet.getImage(index, 0)
126.
127.
128.
129.                #loading walls
130.                wallSheet=Spritesheet("walls.png", 1,
   (0,0,0), 25, 25)
131.                wallList=["wood wall", "copper wall", "tin
   wall", "lead wall", "tungsten wall", "platinum wall", "mithril
   wall", "adamantite wall", "uru wall"]
132.                self.wallImageDict=dict()
```

```
133.
134.                 for index, wall in enumerate(wallList):
135.
    self.wallImageDict[wall]=wallSheet.getImage(index, 0)
136.
137.
138.
139.                 #loading materials
140.                 materialSheet=Spritesheet("materials.png", 1,
    (0,0,0), 30, 24)
141.                 materialList=["copper bar","tin bar","lead
    bar","tungsten bar","platinum bar","mithril bar","adamantite
    bar","uru bar"]
142.                 self.materialImageDict=dict()
143.
144.                 for index, material in
    enumerate(materialList):
145.
    self.materialImageDict[material]=materialSheet.getImage(index,
    0)
146.
147.
148.
149.                 #loading tiles
150.                 tileSheet=Spritesheet("tiles.png", 1,
    (0,0,0), 50, 50)
151.                 tileList=["crafting bench", "anvil",
    "furnace", "chest"]
152.                 self.tileImageDict=dict()
153.
154.                 for index, tile in enumerate(tileList):
155.
    self.tileImageDict[tile]=tileSheet.getImage(index, 0)
156.
157.
158.                 self.itemImageDict={**self.toolImageDict,
    **self.blockImageDict,  **self.wallImageDict,
    **self.materialImageDict, **self.tileImageDict}
159.
160.             def saveWorld(self):
161.                 with
    open(os.path.expanduser(f"{filepathW}{self.worldName}.json"),
    "w") as file:
162.                     #take all the thing i need to save the
    instance of the world: world name, difficulty, timeplayed,
    player pos,health,inv, all the object positions
163.                     save={
164.                         "Name":self.worldName,
```

```
                        "TimePlayed":self.time,
                        "Difficulty":self.difficulty,
                        "Inventory":self.player.inventory,
                        "Health": (self.player.health,
self.player.maxHealth),
                        "Position":
(self.player.rect.x//TILESIZE, self.player.rect.y//TILESIZE),

"backDict":{obj:[self.backDict[obj].x,self.backDict[obj].y,sel
f.backDict[obj].name] for obj in self.backDict},

"gridDict":{obj:[self.gridDict[obj].x,self.gridDict[obj].y,sel
f.gridDict[obj].name] for obj in self.gridDict},

"wallDict":{obj:[self.wallDict[obj].x,self.wallDict[obj].y,sel
f.wallDict[obj].name] for obj in self.wallDict},
                        "treeList":[(obj.x,obj.y) for obj in
self.treeSprites]
                    }
                    json.dump(save, file)

            def loadWorld(self):
                try:
                    with
open(os.path.expanduser(f"{filepathW}{self.worldName}.json"))
as file:
                        save = json.load(file)
                        #load from json dictionary

self.playerInventory=save["Inventory"]
                        self.playerHealth=save["Health"]
                        self.playerPos=save["Position"]
                        backDict=save["backDict"]
                        gridDict=save["gridDict"]
                        wallDict=save["wallDict"]
                        treeList=save["treeList"]

                        self.backDict = dict()
                        self.gridDict = dict()
                        self.wallDict = dict()

                        #for each key in the dictionary I am
accessing the dictionary at that key which gives me the list I
need to pass into the intialisation for each object

                        for sp in backDict:
                            sp=backDict[sp]
```

```
198.                             Background(self, sp[0], sp[1],
    sp[2])
199.
200.                         for sp in gridDict:
201.                             sp=gridDict[sp]
202.
203.                             if sp[2] in self.blockImageDict:
204.                                 Block(self, sp[0], sp[1],
    sp[2])
205.                             elif sp[2] in self.tileImageDict:
206.                                 Tile(self, sp[0], sp[1],
    sp[2])
207.
208.                         for sp in wallDict:
209.                             sp=wallDict[sp]
210.
211.                             Wall(self, sp[0], sp[1], sp[2])
212.
213.                         for sp in treeList:
214.                             Tree(self, sp[0], sp[1])
215.
216.
217.                     #delete overlapping sprites
218.
    pygame.sprite.groupcollide(self.colSprites,
    self.backgroundSprites, False, True)
219.
    pygame.sprite.groupcollide(self.colSprites, self.wallSprites,
    False, True)
220.
    pygame.sprite.groupcollide(self.wallSprites,
    self.backgroundSprites, False, True)
221.
222.                 except:
223.                     #if loading fails
224.                     self.makeWorld()
225.                     self.makePlayer()
226.
227.             def makePlayer(self):
228.
229.                 #intialises the player inventory, health and
    postion. Also gives starting gear
230.                 self.playerInventory=[[i,"empty",0] for i in
    range(43)]
231.
232.                 startingGear=["copper sword", "copper
    pickaxe", "copper axe", "copper hammer"]
233.
```

```
234.                 for index, tool in enumerate(startingGear):
235.                     self.playerInventory[index]=[index, tool,
    1]
236.
237.                 self.playerHealth=(100,100)
238.                 self.playerPos = (20 , 0)
239.
240.
241.             def makeWorld(self):
242.                 #FLOORMAP maps the blocks
243.                 #BACKMAP maps the background
244.                 self.FLOORMAP = [["" for row in
    range(WORLDLENGTH*WIDTH//25)] for column in
    range(WORLDHEIGHT*HEIGHT//25)]
245.                 self.BACKMAP = [["" for row in
    range(WORLDLENGTH*WIDTH//25)] for column in
    range(WORLDHEIGHT*HEIGHT//25)]
246.
247.                 #intialise the 3 tile dictionaries
248.                 self.backDict = dict()
249.                 self.gridDict = dict()
250.                 self.wallDict = dict()
251.
252.                 #dirt to stone 1:2 ratio
253.                 for i,row in enumerate(self.FLOORMAP):
254.                     for j,col in enumerate(row):
255.                         if i<((20*WORLDHEIGHT)//3):
256.                             self.FLOORMAP[i][j]="dirt"
257.                             self.BACKMAP[i][j]="dirt"
258.                         else:
259.                             self.FLOORMAP[i][j]="stone"
260.                             self.BACKMAP[i][j]="stone"
261.
262.                 #draw the random ore veins
263.                 self.squiggle((2,3,3), (8,2), (6,1), (10,5),
    (10,1), (2,1), (5,2), "stone") #stone
264.                 self.squiggle((2,3,3), (6,2), (2,1), (10,5),
    (10,25), (3,2), (10,2), "dirt")#dirt
265.                 self.squiggle((2,3,3), (8,2), (6,1), (10,2),
    (10,10), (2,1), (2,1), "copper")#copper
266.                 self.squiggle((2,3,3), (6,2), (6,1), (10,2),
    (10,10), (2,1), (2,1), "tin")#tin
267.                 self.squiggle((1,2,3), (8,2), (6,1), (10,2),
    (10,10), (2,1), (2,1), "lead")#lead
268.                 self.squiggle((1,2,2), (4,2), (6,1), (10,5),
    (10,25), (2,1), (2,1), "tungsten")#tungsten
269.                 self.squiggle((1,2,2), (4,1), (4,1), (10,5),
    (10,35), (2,1), (2,1), "platinum")#tungsten
```

```
270.                 self.squiggle((1,2), (4,1), (2,1), (10,5),
    (10,40), (2,1), (2,1), "mithril")#mithril
271.
272.                 self.squiggle((2,3,4), (6,2), (2,1), (10,5),
    (10,10), (5,2), (5,2), "air")#caves
273.                  #adamantium and uru are not found naturally,
    found by killing skeleton/zombie
274.
275.                 #make the top section air
276.                 for i,row in enumerate(self.FLOORMAP):
277.                     for j,col in enumerate(row):
278.                         if i<((5*WORLDHEIGHT)//3):
279.                             self.FLOORMAP[i][j]="air"
280.                             self.BACKMAP[i][j]=None
281.
282.         #
    self.drawGrass((0,7),(1,random.uniform(0,0.5)),(1,random.unifo
    rm(0.5,1)),(1.5,random.uniform(5,6)),(1,random.uniform(5,10)),
    (1,random.uniform(2,3)))
283.
    self.drawGrass((0,7),(1,random.uniform(0,0.1)),(1,random.unifo
    rm(0,0.1)),(1.5,random.uniform(0,0.1)),(1,random.uniform(0,0.1
    )),(1,random.uniform(0,0.1)))
284.
285.                 for j,row in enumerate(self.FLOORMAP):
286.                     for i,col in enumerate(row):
287.                         try:
288.                             if self.FLOORMAP[j][i]=="dirt"
    and self.FLOORMAP[j-1][i]=="air":
289.                                 self.FLOORMAP[j][i]="grass"
290.                         except:
291.                             pass
292.
293.
294.                 #world layered with bedrock and blocks are
    placed where there is no air (empty space)
295.                 for j,row in enumerate(self.FLOORMAP):
296.                     for i,col in enumerate(row):
297.
298.                         if i==((WORLDLENGTH*1000)//TILESIZE-
    1) or i==0 or j==((WORLDHEIGHT*500)//TILESIZE-1):
299.                             Block(self, i , j, "bedrock")
300.
301.                         elif self.FLOORMAP[j][i] != "air":
302.                             Block(self, i, j,
    self.FLOORMAP[j][i])
303.
304.                 #background is made from the backmap
```

```
305.                 for j,row in enumerate(self.FLOORMAP):
306.                     for i,col in enumerate(row):
307.                         if self.BACKMAP[j][i]:
308.                             Background(self, i, j,
    self.BACKMAP[j][i])
309.
310.                 #if blocks collide with background then kill
    the backgrounds
311.                 pygame.sprite.groupcollide(self.colSprites,
    self.backgroundSprites, False, True)
312.
313.                 #trees
314.                 for sprite in self.colSprites:
315.                     if sprite.name=="grass":
316.                         x=sprite.rect.x
317.                         y=sprite.rect.top
318.
319.                         #spawn tree ontop of grass
320.                         tree=Tree(self,x,y)
321.                         #kill tree if it colldies with blocks
322.                         if
    pygame.sprite.spritecollideany(tree, self.colSprites):
323.                             if tree:
324.                                 tree.kill()
325.                                 tree=None
326.                         else:
327.                             #kill other trees around that
    tree and respawn the tree in
328.                             pygame.sprite.spritecollide(tree,
    self.treeSprites, True)
329.                             tree=Tree(self,x,y)
330.
331.                         #kill trees are random to get a
    random effect
332.                         if random.uniform(0,1)<=0.3:
333.                             if tree:
334.                                 tree.kill()
335.                                 tree=None
336.
337.
338.             #takes worldPos as a parameter and draws a
    certain block at a certain magnitude around a cetain point
    like a brush does
339.             def brushDraw(self, pos, brush, fill):
340.                 rectCorner=(pos[0]-(brush-1),pos[1]-(brush-
    1))
341.                 for x in range(brush*2-1):
342.                     for y in range(brush*2-1):
```

```
343.
   testPos=(rectCorner[0]+x,rectCorner[1]+y)
344.                            distance=pygame.math.Vector2()
345.                            distance[0]=testPos[0]-pos[0]
346.                            distance[1]=testPos[1]-pos[1]
347.
348.                            if distance.magnitude() <= (brush-
   1):
349.                                try:
350.
   self.FLOORMAP[int(testPos[1])][int(testPos[0])]=fill
351.                                except:
352.                                    pass
353.
354.            #only world pos
355.            def squiggle (self, brushChoice, numberOnX,
   numberOnY, posRange, initPos, widthRange, depthRange, fill):
356.                            # self  (3,4,5), (n, range),(n,
   range),(xRange,yRange), (x,y),  (w, range), (d, range),
   (block)
357.
358.                #random number of ore veins
359.                xNum=numberOnX[0]+random.randint(-
   numberOnX[1],numberOnX[1])
360.                yNum=numberOnY[0]+random.randint(-
   numberOnY[1],numberOnY[1])
361.
362.                #random offset for the starting pos of ore
   veins
363.                xOffset=((WORLDLENGTH*1000//TILESIZE) -
   initPos[0]) // xNum
364.                yOffset=((WORLDHEIGHT*500//TILESIZE) -
   initPos[1]) // yNum
365.
366.                #decide the the random pos
367.                for x in range(xNum):
368.                    for y in range(yNum):
369.
   pos=(initPos[0]+x*(xOffset)+random.randint(-
   posRange[0],posRange[0]),initPos[1]+y*(yOffset)+random.randint
   (-posRange[1],posRange[1]))
370.                            #decide whether to go left or right
371.                            left = random.uniform(0,1)<=0.5
372.                            brush = random.choice(brushChoice)
373.
374.                            #draw that vein instance
375.                            self.drawSquiggle(pos, widthRange,
   depthRange, left, brush, fill)
```

```
376.
377.
378.
379.             #draw vein instance
380.             def drawSquiggle(self, pos, widthRange,
    depthRange, left, brush, fill):
381.                 for depth in
    range(depthRange[0]+random.randint(-
    depthRange[1],depthRange[1])):
382.                     for width in
    range(widthRange[0]+random.randint(-
    widthRange[1],widthRange[1])):
383.                         #draw using brush around block
384.                         self.brushDraw(pos, brush, fill)
385.
386.                         side = random.uniform(0,1)<=0.5
387.                         #choose whether to go to the side or
    diagnoally down
388.
389.                         #based on these choices choose the
    next postion
390.                         if left:
391.                             pos = (pos[0]-1,pos[1]) if side
    else (pos[0]-1,pos[1]+1)
392.                         else:
393.                             pos = (pos[0]+1,pos[1]) if side
    else (pos[0]+1,pos[1]+1)
394.
395.                     left = not(left)
396.
397.
398.             #draw the grass by superposing 5 random sine
    functions
399.             def drawGrass(self, initPos, sine1, sine2, sine3,
    sin4, sin5):
400.                 for x in range(initPos[0],
    ((WORLDLENGTH*1000)//25)):
401.
    y=round((sine1[0]*sin(2*pi*sine1[1]*x))+(sine2[0]*sin(2*pi*sin
    e2[1]*x))+(sine3[0]*sin(2*pi*sine3[1]*x))+(sin4[0]*sin(2*pi*si
    n4[1]*x))+(sin5[0]*sin(2*pi*sin5[1]*x)))+initPos[1]
402.                     self.brushDraw((x,y), 3, "dirt")
403.
404.
405.             #grow a single tree by using the logic the same
    as earlier
406.             def growTree(self):
407.                 for sprite in self.colSprites:
```

```
408.                     if sprite.name=="grass" and
    self.calcDistanceSP(sprite, self.player) > 600:
409.                         x=sprite.rect.x
410.                         y=sprite.rect.top
411.
412.                         tree=Tree(self,x,y)
413.                         if
    pygame.sprite.spritecollideany(tree, self.colSprites):
414.                             if tree:
415.                                 tree.kill()
416.                                 tree=None
417.
418.                         else:
419.                             pygame.sprite.spritecollide(tree,
    self.treeSprites, True)
420.                             tree=Tree(self,x,y)
421.
422.                         break
423.
424.             #if there is empty space above a dirt block then
    it turns into a grass block
425.             def growGrass(self):
426.                 for sprite in self.colSprites:
427.                     try:
428.                         if sprite.name=="dirt" and
    self.gridDict[str((sprite.x,sprite.y-1))]=="air" and
    self.calcDistanceSP(sprite, self.player) > 300:
429.
    self.gridDict[(sprite.x,sprite.y)]==Block(self,sprite.x,sprite
    .y,"grass")
430.                             sprite.kill()
431.                     except:
432.                         pass
433.
434.             #spawn a single enemy depending on the time of
    day
435.             def spawnEnemy(self):
436.                 enemyDay=[Slime]
437.                 enemyNight=[Zombie,Skeleton]
438.
439.                 if self.lux <0.3:
440.                     enemy=random.choice(enemyNight)(self,
    self.player.rect.x//TILESIZE+random.randint(20,32) * (-1 if
    random.uniform(0,1) <=0.5 else 1), 0)
441.                 else:
442.                     enemy=random.choice(enemyDay)(self,
    self.player.rect.x//TILESIZE+random.randint(20,32) * (-1 if
    random.uniform(0,1) <=0.5 else 1), 0)
```

```
443.
444.
445.             def events(self):
446.
447.                 for event in pygame.event.get():
448.
449.                     #save word on quit
450.                     if event.type == pygame.QUIT:
451.                         self.saveWorld()
452.                         menu(self.screen)
453.
454.                     #on keypress
455.                     if event.type==KEYDOWN:
456.                         #if a num key is pressed
457.                         if event.key in
    [K_1,K_2,K_3,K_4,K_5,K_6,K_7,K_8,K_9,K_0]:
458.                             #change the hotbar selection
459.
    self.inv.changeSelected(event.key-49)
460.
461.                         #jump
462.                         if self.player.grounded and event.key
    == K_SPACE and not self.noclip:
463.                             self.player.vel.y = -
    self.player.jumpSpeed
464.                             self.player.grounded = False
465.
466.                         #toggle the inventory and refresh
    crafting
467.                         if event.key == K_ESCAPE:
468.                             self.inv.toggleInv()
469.
470.                             if self.inv.inventoryVisible:
471.                                 self.inv.updateCrafting()
472.
473.                     if event.type==MOUSEBUTTONDOWN:
474.                         #use a tool/weapon/item/place a block
475.                         if not self.hudEvent(event):
476.                             if event.button==1:
477.                                 self.player.use(event)
478.
479.                        #scroll the hotbar selection
480.                         if event.button==4:
481.                             self.inv.scroll(0)
482.
483.                         if event.button==5:
484.                             self.inv.scroll(1)
485.
```

```
486.
487.
488.
489.             def hudEvent(self, event):
490.                 hudClicked=False
491.                 if event.button==1:
492.                     #if you left click on a inventory object
493.                     for sp in self.hudSprites:
494.                         if sp.rect.collidepoint(event.pos):
495.
496.                             if
    sp.__class__.__name__=="invUnit":
497.                                 hudClicked=True
498.
499.                                 #if that object is an empty
    unit
500.                                 if sp.itemName=="empty":
501.                                     #and you are holding
    something in hand
502.                                     if self.hand:
503.
    self.player.invChange(sp.index, self.hand.itemName,
    self.hand.itemCount)
504.                                         self.hand.kill()
505.                                         self.hand=None
506.                                         #deposit
507.
508.                                 else:
509.                                     #if the object is not an
    empty unit
510.                                     if self.hand:
511.                                         #and you are holding
    something in hand
512.                                         if
    self.hand.itemName==sp.itemName:
513.                                             #if it is an item
    of the same name then try to combine the items
514.                                             if
    self.player.invChange(sp.index, self.hand.itemName,
    self.hand.itemCount+sp.itemCount):
515.
    self.hand.kill()
516.
    self.hand=None
517.                                             else:
518.                                                 #if this is
    not possible then try to fill the inv slot as much as possible
```

```
519.                                                 if
   sp.itemCount==32:
520.                                                     #if the
   inventory slot is full then swap your hand and inv slot
521.
   self.player.invChange(sp.index, self.hand.itemName,
   self.hand.itemCount)
522.
   self.hand.kill()
523.
   self.hand=Hand(self, sp)
524.                                                 else:
525.
   sp.itemCount=self.hand.itemCount+sp.itemCount -
   self.player.maxItemCount
526.
   self.player.invChange(sp.index, self.hand.itemName,
   self.player.maxItemCount)
527.
   self.hand.kill()
528.
   self.hand=Hand(self, sp)
529.
530.                                             #otherwise swap hand
   and inv slot
531.                                             else:
532.
   self.player.invChange(sp.index, self.hand.itemName,
   self.hand.itemCount)
533.                                                 self.hand.kill()
534.
   self.hand=Hand(self, sp)
535.
536.                                         #otherwise pick
   everything to hand
537.                                         else:
538.
   self.player.invChange(sp.index,"empty",0)
539.                                             self.hand=Hand(self,
   sp)
540.
541.                                 elif
   sp.__class__.__name__=="craftUnit":
542.                                     hudClicked=True
543.                                     #if instead you left click on
   the crafting menu and you click on a non empty cell then it
   will allow you to craft said item
544.                                     if sp.itemName=="empty":
```

```
                                    pass

                                else:
self.player.craft(sp.itemName)
                    #place item back where it was found
                    if hudClicked==False and self.hand:
intialCount=self.player.inventory[self.hand.index][2]
self.player.invChange(self.hand.index,self.hand.itemName,intialCount+self.hand.itemCount)
                        self.hand.kill()
                        self.hand=None
                        hudClicked=True

                #right click on inventory
                if event.button==3:
                    hudClicked=False
                    for sp in self.hudSprites:
                        if sp.rect.collidepoint(event.pos):
                            if sp.__class__.__name__=="invUnit":
                                hudClicked=True
                                #if you right click on empty unit
                                if sp.itemName=="empty":

                                    if self.hand:
self.player.invChange(sp.index,self.hand.itemName,1)
                                        self.hand.decrement()
                                        #drop one item into the unit from hand

                                elif self.hand:
                                    #still drop a unit
                                    if self.hand.itemName==sp.itemName:
                                        sp.itemCount+=1
self.player.invChange(sp.index,sp.itemName,sp.itemCount)
                                        self.hand.decrement()

                                else:
                                    #swap
```

```
582.
    self.player.invChange(sp.index,self.hand.itemName,self.hand.it
    emCount)
583.                                             self.hand.kill()
584.                                             self.hand=Hand(self,
    sp)
585.
586.                                     else:
587.                                         #pick up an item
588.                                         sp.itemCount-=1
589.
    self.player.invChange(sp.index,sp.itemName,sp.itemCount)
590.                                         sp.itemCount=1
591.                                         self.hand=Hand(self, sp)
592.
593.                                 #no nothing for crafting
594.                                 elif
    sp.__class__.__name__=="craftUnit":
595.                                     hudClicked=True
596.
597.                         #drop item on right click
598.                         if hudClicked==False and self.hand:
599.                             for item in
    range(self.hand.itemCount):
600.                                 Item(self, self.hand.rect.x-
    self.off[0], self.hand.rect.y-self.off[1], self.hand.itemName)
601.                             self.hand.kill()
602.                             self.hand=None
603.                             hudClicked=True
604.
605.
606.                 return hudClicked
607.             #damage a block if there is a block at mouse Pos
    and it is within the players range
608.             def destroyBlock(self, event, tool):
609.                 pos = event.pos - self.off
610.                 worldPos = (pos[0] //TILESIZE,pos[1]
    //TILESIZE)
611.
612.                 if self.calcDistance(pos, self.player) <
    self.player.range:
613.                     for sp in self.colSprites:
614.                         if (sp.x,sp.y) == worldPos:
615.                             sp.damage(tool.power)
616.                             break
617.
618.             #damage a tree if there is a block at mouse Pos
    and it is within the players range
```

```
619.             def destroyTree(self, event, tool):
620.                 pos = event.pos - self.off
621.                 worldPos = (pos[0] //TILESIZE,pos[1]
   //TILESIZE)
622.
623.                 if self.calcDistance(pos, self.player) <
   self.player.range:
624.                     for sp in self.treeSprites:
625.                         posInMask=(pos[0] - sp.rect.x, pos[1]
   - sp.rect.y)
626.                         if sp.rect.collidepoint(pos) and
   sp.mask.get_at((int(posInMask[0]),int(posInMask[1]))):
627.                             sp.damage(tool.power)
628.
629.                             break
630.
631.             #damage a wall if there is a block at mouse Pos
   and it is within the players range
632.             def destroyWall(self, event, tool):
633.                 pos = event.pos - self.off
634.                 worldPos = (pos[0] //TILESIZE,pos[1]
   //TILESIZE)
635.
636.                 if self.calcDistance(pos, self.player) <
   self.player.range:
637.                     for sp in self.wallSprites:
638.                         if sp.rect.collidepoint(pos):
639.                             sp.damage(tool.power)
640.
641.                             break
642.
643.             #place a tile if there is a non solid block at
   mouse Pos (or no block) and it is within the players range
644.             #decrement inventory if you place a block
645.             def place(self, event):
646.                 pos = event.pos - self.off
647.                 worldPos = (pos[0] //TILESIZE,pos[1]
   //TILESIZE)
648.
649.                 colliding=False
650.                 for sprite in self.colSprites:
651.                     if sprite.rect.collidepoint(pos):
652.                         colliding=True
653.
654.                 for sprite in self.treeSprites:
655.                     if sprite.rect.collidepoint(pos):
656.                         colliding=True
657.
```

```
658.                 if self.calcDistance(pos, self.player) <
    self.player.range and not colliding:
659.                     accessed=False
660.
    unit=self.player.inventory[self.player.selectedIndex]
661.
662.                     if unit[1] in self.blockImageDict.keys():
663.                         Block(self,*worldPos,unit[1])
664.
665.                         if str((worldPos[0],worldPos[1])) in
    self.wallDict:
666.
    self.wallDict[str((worldPos[0],worldPos[1]))].kill
667.
668.                         elif str((worldPos[0],worldPos[1]))
    in self.backDict:
669.
    self.backDict[str((worldPos[0],worldPos[1]))].kill
670.
671.                         accessed=True
672.
673.                     elif unit[1] in
    self.wallImageDict.keys():
674.                         Wall(self, *worldPos, unit[1])
675.
676.                         if str((worldPos[0],worldPos[1])) in
    self.backDict:
677.
    self.backDict[str((worldPos[0],worldPos[1]))].kill
678.
679.                         accessed=True
680.
681.                     elif unit[1] in
    self.tileImageDict.keys():
682.
683.                         Tile(self,worldPos[0],worldPos[1]-1,
    unit[1])
684.
685.                         accessed=True
686.
687.
688.                     if accessed:
689.                         unit[2]-=1
690.                         self.player.invChange(*unit)
691.
692.
693.
```

```
694.             #calculates the distance between a sprite and
   mouse position
695.             def calcDistance(self, position, sprite):
696.                 distance=pygame.math.Vector2()
697.                 distance[0]=(position[0])-
   (sprite.rect.centerx)
698.                 distance[1]=(position[1])-
   (sprite.rect.centery)
699.                 return distance.magnitude()
700.
701.             #calculates the distance between a sprite and
   another sprite
702.             def calcDistanceSP(self, sprite1, sprite2):
703.                 distance=pygame.math.Vector2()
704.                 distance[0]=(sprite1.rect.centerx)-
   (sprite2.rect.centerx)
705.                 distance[1]=(sprite1.rect.centery)-
   (sprite2.rect.centery)
706.
707.                 return distance.magnitude()
708.
709.             #update
710.             def update(self):
711.                 #update the time
712.                 self.time = time() - self.initTime +
   self.startTime
713.                 #update the colour of the sky
714.
   self.lux=0.5*cos((2*pi)/self.timePeriod*self.time)+0.5
715.                 # if there are too few trees then grow one
716.                 if len(self.treeSprites) < 3:
717.                     self.growTree()
718.
719.                 #on tick grow trees/grass/enemy
720.                 if self.tick==0:
721.                     self.tick=500
722.                     self.growTree()
723.                     self.growGrass()
724.                     if len(self.enemySprites)<3:
725.                         self.spawnEnemy()
726.
727.                 if self.difficulty != "Casual" and
   self.tick==250 and len(self.enemySprites)<3:
728.                     print("spawmn")
729.                     self.spawnEnemy()
730.
731.                 #decrement tick
732.                 self.tick-=1
```

```
733.
734.                 #update the HUD
735.                 self.healthBar.displayMessage()
736.                 if self.hand:
737.                     self.hand.update()
738.                 if self.player.healthChange:
739.                     self.healthBar.update(self.player.health,
    self.player.maxHealth)
740.                     self.player.healthChange=False
741.                 if self.player.inventoryChange:
742.                     self.inv.update(self.player.inventory)
743.                     self.player.inventoryChange=False
744.
745.                 #update the camera sprite groups
746.                 ...
747.                 self.camSprites.update()
748.                 self.personSprites.update()
749.                 self.backSprites.update()
750.
751.
752.                 #update the FPS and delta time
753.                 self.clock.tick(self.FPS)
754.                 now = time()
755.                 self.dt = now - self.prevTime
756.                 self.prevTime = now
757.
758.
759.             #fill find all blocks in a 5 block radius and
    call update darkness
760.             def updateDarknessAR(self, block):
761.                 brush=5
762.                 pos=(block.x,block.y)
763.                 rectCorner=(pos[0]-(brush-1),pos[1]-(brush-
    1))
764.                 for x in range(brush*2-1):
765.                     for y in range(brush*2-1):
766.
    testPos=(rectCorner[0]+x,rectCorner[1]+y)
767.                         distance=pygame.math.Vector2()
768.                         distance[0]=testPos[0]-pos[0]
769.                         distance[1]=testPos[1]-pos[1]
770.
771.                         if distance.magnitude() <= (brush-1)
    and str((testPos[0],testPos[1])) in self.gridDict:
772.
    self.updateDarkness(self.gridDict[str((testPos[0],testPos[1]))
    ])
773.
```

```
774.
775.            #will determine the level of darkness that a
   block needs by the number of other blocks around it.
776.            def updateDarkness(self, block):
777.                if block:
778.
779.                    spaceFound=False
780.                    brush=1
781.                    pos=(block.x,block.y)
782.                    while not spaceFound and brush < 5:
783.                        brush+=1
784.                        rectCorner=(pos[0]-(brush-1),pos[1]-
   (brush-1))
785.                        for x in range(brush*2-1):
786.                            for y in range(brush*2-1):
787.
   testPos=(rectCorner[0]+x,rectCorner[1]+y)
788.
   distance=pygame.math.Vector2()
789.                                    distance[0]=testPos[0]-
   pos[0]
790.                                    distance[1]=testPos[1]-
   pos[1]
791.
792.                                    if (brush-2) <
   distance.magnitude() <= (brush-1):
793.                                        if
   str((testPos[0],testPos[1])) in self.gridDict:
794.                                            if
   self.gridDict[str((testPos[0],testPos[1]))].name=="bedrock":
795.                                                brush=5
796.                                        else:
797.                                            spaceFound=True
798.
799.                    block.updateDarkness((brush-2))
800.
801.
802.
803.            def draw(self):
804.                ...
805.                #draw the sprite grroups
806.
   self.screen.fill(self.colorFader.mix(self.lux))
807.                self.backSprites.customDraw()
808.                self.camSprites.customDraw()
809.                self.personSprites.customDraw()
810.                self.hudSprites.customDraw()
811.
```

```
812.
813.                 #draw the FPS
814.                 text = self.font.render("FPS:
   "+str(self.clock.get_fps())[:5],1,(255, 226, 18))
815.                 self.screen.blit(text, (700,20))
816.
817.
818.                 #update the display
819.                 pygame.display.update()
820.
821.             def main(self):  # this is the main game loop
822.                 while True:
823.                     self.events()
824.                     self.update()
825.                     self.draw()
826.
827.         #hand class which is for holding items temeporarily
   when you need to shift items around the inventory
828.         class Hand(pygame.sprite.Sprite):
829.             def __init__(self, game, sprite):
830.                 pos = pygame.mouse.get_pos()
831.                 self.game=game
832.                 self.groups = self.game.hudSprites
833.                 pygame.sprite.Sprite.__init__(self,
   self.groups)
834.                 #intialise groups
835.
836.                 #takes on characteristics of invUnit
837.                 self.index=sprite.index
838.                 self.itemName=sprite.itemName
839.                 self.itemCount=sprite.itemCount
840.
   self.itemImage=self.game.itemImageDict[self.itemName]
841.
842.                 #adusts the size of the image so it fits
   within 25x25
843.                 sizeChange=False
844.                 size=self.itemImage.get_size()
845.                 sfx=1
846.                 sfy=1
847.                 if size[0] > TILESIZE:
848.                     sfx=size[0]//TILESIZE
849.                     sizeChange=True
850.                 if size[1] > TILESIZE:
851.                     sfy=size[1]//TILESIZE
852.                     sizeChange=True
853.
854.
```

```
                if sizeChange:
                    sf=max(sfy,sfx)

self.itemImage=pygame.transform.scale(self.itemImage,
(size[0]//sf, size[1]//sf))
                    self.itemImage.set_colorkey((0,0,0))


                #blit the item image and item count to
surface

self.image=pygame.Surface((50,50)).convert_alpha()
                self.image.set_colorkey((0,0,0))
                self.image.blit(self.itemImage, (10,10))

self.image.blit(self.game.font.render(str(self.itemCount), 1,
(255,255,255)).convert_alpha(), (10,35))

                #rect
                self.rect=self.image.get_rect()
                self.rect.x=pos[0]
                self.rect.y=pos[1]

            def update(self):
                #stick it to the mouse position
                pos = pygame.mouse.get_pos()
                self.rect.x=pos[0]
                self.rect.y=pos[1]

            def decrement(self):
                #decrease item count by one and update image
                self.itemCount-=1

self.image=pygame.Surface((50,50)).convert_alpha()
                self.image.set_colorkey((0,0,0))
                self.image.blit(self.itemImage, (10,10))

self.image.blit(self.game.font.render(str(self.itemCount), 1,
(255,255,255)).convert_alpha(), (10,35))
                if self.itemCount==0:
                    self.game.hand.kill()
                    self.game.hand=None

        #will blit text for a small amount of time - is used
as an interrupt message
        class Text(pygame.sprite.Sprite):
            def __init__(self, game, text, x, y):
                self.game = game
```

```
893.                 self.groups = (self.game.hudSprites)
894.                 pygame.sprite.Sprite.__init__(self,
    self.groups)
895.                 self.text=text
896.
897.
    self.image=self.game.font.render(str(self.text), 1,
    (255,226,18)).convert_alpha()
898.                 self.rect=self.image.get_rect()
899.                 self.rect.x=x*TILESIZE
900.                 self.rect.y=y*TILESIZE
901.
902.         #healthbar hud class
903.         class HealthBar(pygame.sprite.Sprite):
904.             def __init__(self, game, x, y, scale, maxHealth,
    health):
905.                 self.game = game
906.                 self.groups = (self.game.hudSprites)
907.                 pygame.sprite.Sprite.__init__(self,
    self.groups)
908.
909.                 self.colour=(0,0,0)
910.                 self.scale=scale
911.                 #load the maxHealth and health
912.                 self.maxHealth=maxHealth
913.                 self.health=health
914.                 #load the images from sprite sheet
915.                 self.healthSheet=Spritesheet("health.png",
    self.scale, self.colour, 26, 24)
916.                 self.healthList=[self.healthSheet.getImage(i,
    0) for i in range(21)]
917.                 self.healthMessage=None
918.
919.                 #print onto surface while enlarging it to
    scale
920.
    self.image=pygame.Surface(((self.maxHealth/20)*26*self.scale,
    24*self.scale)).convert_alpha()
921.                 self.image.set_colorkey(self.colour)
922.                 self.rect=self.image.get_rect()
923.                 self.rect.x=x
924.                 self.rect.y=y
925.                 #call update
926.                 self.updateHealth()
927.
928.             #passes player health into the HUD for update
929.             def update(self, health, maxHealth):
930.                 self.health=health
```

```
931.                 self.maxHealth=maxHealth
932.
933.                 self.updateHealth()
934.
935.             #displays exact health if you hover over the rect
936.             def displayMessage(self):
937.                 pos = pygame.mouse.get_pos()
938.
939.                 if self.rect.collidepoint(pos):
940.                     if self.healthMessage:
941.                         self.healthMessage.kill()
942.
943.                     self.healthMessage=Text(self.game,
    f"{str(int(self.health))}/{str(self.maxHealth)}", pos[0]+20,
    pos[1]+10)
944.
945.                 else:
946.                     if self.healthMessage:
947.                         self.healthMessage.kill()
948.
949.             #updates the number of hearts in the health bar
    key
950.             def updateHealth(self):
951.                 health=int(self.health)
952.
953.                 for i in range(int(self.maxHealth/20)):
954.                     if health>=20:
955.                         heart=20
956.                         health-=20
957.                     elif health>0:
958.                         heart=health
959.                         health=0
960.                     else:
961.                         heart=0
962.
963.                     self.image.blit(self.healthList[heart],
    (26*i*self.scale, 0, 26*self.scale,24*self.scale))
964.
965.
966.
967.         #inventory unit
968.         class invUnit(pygame.sprite.Sprite):
969.             def __init__(self, game, index, x, y):
970.                 self.game = game
971.                 self.groups = (self.game.hudSprites)
972.                 pygame.sprite.Sprite.__init__(self,
    self.groups)
973.
```

```
974.                 #loads base and selected image
975.
   self.invBase=pygame.image.load(os.path.expanduser(filepathG +
   "invunit.png")).convert_alpha()
976.
   self.invSelected=pygame.image.load(os.path.expanduser(filepath
   G + "invunitSelected.png")).convert_alpha()
977.                 self.font =
   pygame.font.SysFont("AndyBold",30)
978.
979.                 #loads image/rect and position
980.                 self.selected=False
981.                 self.image =
   pygame.Surface((50,50)).convert_alpha()
982.                 self.rect = self.image.get_rect()
983.                 self.rect.x = x
984.                 self.rect.y = y
985.
986.                 #starts as an empty unit
987.                 self.index=index
988.                 self.itemName="empty"
989.                 self.itemCount=0
990.
991.             #function to allow hotbar selected to change
992.             def updateStatus(self, selected):
993.                 self.selected=selected
994.
995.             #allows the invUnit to change
996.             def changeItem(self, index, item, count):
997.                 self.index=index
998.                 self.itemName=item
999.                 self.itemCount=count
1000.
1001.            #draw the inventory unit
1002.            def draw(self):
1003.                self.game.hudSprites.add(self)
1004.                self.image =
   pygame.Surface((50,50)).convert_alpha()
1005.                self.image.set_colorkey((0,0,0))
1006.                self.image.blit((self.invSelected if
   self.selected else self.invBase), (0,0))
1007.
   self.image.blit(self.font.render(str(self.index+1), 1,
   (255,255,255)).convert_alpha(), (0,0))
1008.
1009.                if self.itemCount > 0:
1010.
   image=self.game.itemImageDict[self.itemName]
```

```
1011.                     #transform the scale so it fits
1012.                     size=self.image.get_size()
1013.                     if size[0] > TILESIZE:
1014.                         sfx=size[0]//TILESIZE
1015.                     if size[1] > TILESIZE:
1016.                         sfy=size[1]//TILESIZE
1017.                     sf=max(sfy,sfx)
1018.                     image=pygame.transform.scale(image,
   (size[0]//sf, size[1]//sf))
1019.                     image.set_colorkey((0,0,0))
1020.
1021.                     #blit the item image on the inventory
   unit
1022.                     self.image.blit(image, (10,10))
1023.
   self.image.blit(self.font.render(str(self.itemCount), 1,
   (255,255,255)).convert_alpha(), (10,35))
1024.
1025.             #clear that unit
1026.             def clear(self):
1027.                 self.kill()
1028.
1029.             def __str__(self):
1030.                     return
   [self.index,self.itemName,self.itemCount]
1031.
1032.         #crafting unit
1033.         class craftUnit(pygame.sprite.Sprite):
1034.             def __init__(self, game, index, x, y):
1035.                 #init group and call inheritance
1036.                 self.game = game
1037.                 self.groups = (self.game.hudSprites)
1038.                 pygame.sprite.Sprite.__init__(self,
   self.groups)
1039.
1040.                 #init image
1041.
   self.unit=pygame.image.load(os.path.expanduser(filepathG +
   "craftunit.png")).convert_alpha()
1042.                 self.font =
   pygame.font.SysFont("AndyBold",30)
1043.
1044.                 self.image =
   pygame.Surface((50,50)).convert_alpha()
1045.                 self.rect = self.image.get_rect()
1046.                 self.rect.x = x
1047.                 self.rect.y = y
1048.
```

```
1049.                 #empty unit
1050.                 self.index=index
1051.                 self.itemName="empty"
1052.                 self.itemCount=0
1053.
1054.             #allow for changing
1055.             def changeItem(self, index, item, count):
1056.                 self.index=index
1057.                 self.itemName=item
1058.                 self.itemCount=count
1059.
1060.             #draw to the crafting unit
1061.             def draw(self):
1062.                 self.game.hudSprites.add(self)
1063.                 self.image =
    pygame.Surface((50,50)).convert_alpha()
1064.                 self.image.set_colorkey((0,0,0))
1065.                 self.image.blit((self.unit), (0,0))
1066.
    self.image.blit(self.font.render(str(self.index+1), 1,
    (255,255,255)).convert_alpha(), (0,0))
1067.
1068.                 if self.itemCount > 0:
1069.
    image=self.game.itemImageDict[self.itemName]
1070.                     #tranform scale
1071.                     size=self.image.get_size()
1072.                     if size[0] > TILESIZE:
1073.                         sfx=size[0]//TILESIZE
1074.                     if size[1] > TILESIZE:
1075.                         sfy=size[1]//TILESIZE
1076.                     sf=max(sfy,sfx)
1077.                     image=pygame.transform.scale(image,
    (size[0]//sf, size[1]//sf))
1078.                     image.set_colorkey((0,0,0))
1079.
1080.                     self.image.blit(image, (10,10))
1081.
    self.image.blit(self.font.render(str(self.itemCount), 1,
    (255,255,255)).convert_alpha(), (10,35))
1082.
1083.             def clear(self):
1084.                 self.kill()
1085.
1086.             def __str__(self):
1087.                     return f"{self.index}, {self.itemName} ,
    {self.itemCount}"
1088.
```

```
1089.
1090.        class Inventory(pygame.sprite.Sprite):
1091.            def __init__(self, game):
1092.                self.game = game
1093.                #create hotbar
1094.                self.hotbar=[invUnit(game, i, 5+60*i, 50) for
   i in range(10)]
1095.                self.hotbar[0].selected=True
1096.
1097.                #create the storage and armour sections
1098.                self.storage=[invUnit(game, i+10,
   5+60*(i%10), 150 + 60*(i//10)) for i in range(30)]
1099.                self.armour=[invUnit(game, i+40, 800, 150 +
   60*i) for i in range(3)]
1100.                self.inventory=self.storage+self.armour
1101.                #create the crafting section
1102.                self.crafting=[craftUnit(game, i,
   5+60*(i%15), 350 + 60*(i//15)) for i in range(30)]
1103.                self.inventoryVisible=False
1104.
1105.                #load the recipe dictionary
1106.                self.recipeDict=dict()
1107.                with
   open(os.path.expanduser(f"{filepathJ}recipe.json")) as f:
1108.                    data=json.load(f)
1109.
1110.                self.recipeDict=data
1111.
1112.                self.draw()
1113.
1114.        #     def updateHotbar(self):
1115.        #         self.hotbarImage=pygame.Surface(590,
   50).convert_alpha()
1116.        #         for item in self.hotbar:
1117.        #             item.draw()
1118.        #             self.hotbarImage.blit(item.image,
   (item.rect.x, item.rect.y))
1119.
1120.            # if esc pressed and inventory closed then open
   it, if the inventory is open then close it
1121.            def toggleInv(self):
1122.                if self.inventoryVisible:
1123.                    self.inventoryVisible=False
1124.                else:
1125.                    self.inventoryVisible=True
1126.
1127.                self.draw()
1128.
```

```
1129.           #scroll the hotbar up and down, making sure it
    wraps around the edge by using %
1130.           def scroll(self, up):
1131.               for index, item in enumerate(self.hotbar):
1132.                   if item.selected:
1133.                       item.selected=False
1134.                       if up:
1135.
    newIndex=(index+1)%len(self.hotbar)
1136.
    self.hotbar[newIndex].selected=True
1137.                       else:
1138.                           newIndex=(index-
    1)%len(self.hotbar)
1139.
    self.hotbar[newIndex].selected=True
1140.
1141.
    self.game.player.selectedIndex=newIndex
1142.                       break
1143.
1144.               self.draw()
1145.
1146.           #changes what hotbar index is selected by using
    the num keys
1147.           def changeSelected(self, key):
1148.               for i in range(len(self.hotbar)):
1149.                   self.hotbar[i].selected=False
1150.
1151.               self.hotbar[key].selected=True
1152.
1153.               self.game.player.selectedIndex=key
1154.
1155.               self.draw()
1156.
1157.
1158.           #passes the player invetory in the inventory HUD
1159.           def update(self, playerInv):
1160.
1161.               #the first 10 are hotbar units the last 30
    are storage units
1162.               for unit in playerInv:
1163.                   if unit[0]<10:
1164.
    self.hotbar[unit[0]].changeItem(unit[0],unit[1],unit[2])
1165.
1166.                   elif unit[0]<40:
```

```
                        self.inventory[unit[0]-10].changeItem(unit[0],unit[1],unit[2])

                self.draw()

            #updates the crafting menu on refresh
            def updateCrafting(self):
                self.clearCrafting()

                #depending on what tile a player is colliding with
                if self.game.player.touching:
                    self.tile=self.game.player.touching

                else:
                    self.tile="nothing"

                #accesses the recipeDict from the json file and looks under the tile which the player is colliding with
                for key in self.recipeDict[self.tile]:
                    #for every recipe which is in this part of the dictionary
                    ingList=list(self.recipeDict[self.tile][key].keys())
                    itemCounts=[]
                    #gets the list of ingredients and the list of the amound of those ingredients
                    for ing in ingList:
                        materialCount=0
                        for unit in self.game.player.inventory:
                            if ing==unit[1]:
                                materialCount+=unit[2]

                        #rounds up all of the relevant matrials
                        itemCounts.append(materialCount//self.recipeDict[self.tile][key][ing])
        #                    break
                    #if in creative ignore material costs
                    if self.game.difficulty !="Creative":
                        numItems=min(itemCounts)
                        if numItems>=1:
                            self.addRecipe(key,numItems)

                    #add the recipe to the crafting menu depending on the number of items the player can make
```

```
                    else:
                        self.addRecipe(key,float("inf"))


                #update the inventory
                self.draw()

            #adds the recipe to the crafting menu if more than 1 can be made
            def addRecipe(self,key,numItems):
                for unit in self.crafting:
                    if unit.itemCount==0:
                        self.crafting[unit.index].changeItem(unit.index,key,numItems)
                        break

            #clears the crafting window
            def clearCrafting(self):
                for unit in self.crafting:
                    self.crafting[unit.index].changeItem(unit.index,"empty",0)

            #redrwas all of the units in the inventory
            def draw(self):
                if self.inventoryVisible:
                    for item in self.inventory:
                        item.draw()
                    for item in self.crafting:
                        item.draw()
                else:
                    for item in self.inventory:
                        item.clear()
                    for item in self.crafting:
                        item.clear()

                for item in self.hotbar:
                    item.draw()

        #tool class
        class Tool(pygame.sprite.Sprite):
            def __init__(self, game, tool):
                self.game=game
                self.groups = (self.game.camSprites)
                pygame.sprite.Sprite.__init__(self, self.groups)

                #intialises the sprites hit as none
                self.spritesHit=[]
```

```
1248.                 #gets the tool name
1249.                 self.tool=tool
1250.                 #gets the image and reverse image of the tool
1251.
1252.
   self.initImage=self.game.toolImageDict[self.tool]
1253.
   self.reverseImage=pygame.transform.flip(self.initImage, True,
   False)
1254.
1255.                 #gets the stats of tool from dictionary
1256.
   self.speed=self.game.toolStatDict[self.tool]["speed"]
1257.
   self.damage=self.game.toolStatDict[self.tool]["damage"]
1258.
   self.type=self.game.toolStatDict[self.tool]["type"]
1259.
   self.power=self.game.toolStatDict[self.tool]["power"]
1260.
1261.                 #if the tool is a high level sword then allow
   it to shoot projectiles
1262.                 if self.tool.split(" ")[1]=="sword" and
   self.tool.split(" ")[0] in ["mithril", "adamantite", "uru"]:
1263.                     Fireball(self.game,
   self.game.player.rect.x, self.game.player.rect.y, -500 if
   self.game.player.lookingLeft else 500, 0, True, self.power)
1264.
1265.                 #init image/rect and position
1266.
   self.image=pygame.Surface((50,50)).convert_alpha()
1267.                 self.image.set_colorkey((0,0,0))
1268.
   self.mask=pygame.mask.from_surface(self.image)
1269.                 self.rect=self.image.get_rect()
1270.                 self.center=self.game.player.rect.center
1271.
1272.
1273.             def updateFrame(self, frame, left):
1274.                 self.center=self.game.player.rect.center
1275.
1276.                 #stick the tool onto the player using
   animation, depending on whether the player is facing right or
   left
1277.                 if left:
1278.                     if frame==1:
```

```
1279.
   self.image=pygame.transform.rotate(self.reverseImage,
   0).convert_alpha()
1280.                         self.rect.center=(self.center[0]-30,
   self.center[1]-35)
1281.
   self.mask=pygame.mask.from_surface(self.image)
1282.                     if frame==2:
1283.
   self.image=pygame.transform.rotate(self.reverseImage,
   30).convert_alpha()
1284.                         self.rect.center=(self.center[0]-50,
   self.center[1]-20)
1285.
   self.mask=pygame.mask.from_surface(self.image)
1286.                     if frame==3:
1287.
   self.image=pygame.transform.rotate(self.reverseImage,
   50).convert_alpha()
1288.                         self.rect.center=(self.center[0]-50,
   self.center[1]+2)
1289.
   self.mask=pygame.mask.from_surface(self.image)
1290.                     if frame==4:
1291.
   self.image=pygame.transform.rotate(self.reverseImage,
   80).convert_alpha()
1292.                         self.rect.center=(self.center[0]-37,
   self.center[1]+30)
1293.
   self.mask=pygame.mask.from_surface(self.image)
1294.
1295.                 else:
1296.                     if frame==1:
1297.
   self.image=pygame.transform.rotate(self.initImage, -
   5).convert_alpha()
1298.                         self.rect.center=(self.center[0]+30,
   self.center[1]-35)
1299.
   self.mask=pygame.mask.from_surface(self.image)
1300.                     if frame==2:
1301.
   self.image=pygame.transform.rotate(self.initImage, -
   35).convert_alpha()
1302.                         self.rect.center=(self.center[0]+30,
   self.center[1]-17)
```

```
1303.
   self.mask=pygame.mask.from_surface(self.image)
1304.                     if frame==3:
1305.
   self.image=pygame.transform.rotate(self.initImage, -
   50).convert_alpha()
1306.                         self.rect.center=(self.center[0]+30,
   self.center[1]+1)
1307.
   self.mask=pygame.mask.from_surface(self.image)
1308.                     if frame==4:
1309.
   self.image=pygame.transform.rotate(self.initImage, -
   70).convert_alpha()
1310.                         self.rect.center=(self.center[0]+25,
   self.center[1]+20)
1311.
   self.mask=pygame.mask.from_surface(self.image)
1312.
1313.                 self.checkCollision()
1314.
1315.             #check a collision between the tool and
   enemySprites, if there is one then deal damage to that enemy
   based on the tools damage and knock the enemy back based on
   which direction the player is looking
1316.             def checkCollision(self):
1317.                 hits = pygame.sprite.spritecollide(self,
   self.game.enemySprites, False)
1318.                 for hit in hits:
1319.                     if hit not in self.spritesHit:
1320.
1321.                         if self.game.player.lookingLeft:
1322.                             hit.takeDamage(self.damage,
   "left")
1323.                         else:
1324.                             hit.takeDamage(self.damage,
   "right")
1325.
1326.                         self.spritesHit.append(hit)
1327.
1328.         #player
1329.         class Player(pygame.sprite.Sprite):
1330.             def __init__(self, game, x, y, inventory, health,
   maxHealth): # this x,y coord is a world coord (not pixels)
1331.                 self.game = game
1332.                 self.groups = (self.game.personSprites)
1333.                 pygame.sprite.Sprite.__init__(self,
   self.groups)
```

```
1334.
1335.                 #initialise constants and variables
1336.                 self.vel = pygame.math.Vector2()
1337.
1338.                 if self.game.noclip:
1339.                     self.speed=1000
1340.                 else:
1341.                     self.speed = 200
1342.
1343.
1344.                 self.jumpSpeed=800
1345.                 self.grav = 2500  #gravity a player
     experiences
1346.                 self.range=100  #how far a player can reach
1347.                 self.grounded = True  #on the ground
1348.                 self.lookingLeft=True
1349.                 self.walking=False
1350.                 self.using=False
1351.                 self.jumpHeight=0
1352.
1353.                 self.maxHealth=maxHealth #init maxHealth
1354.                 self.health=(health if health > 0 else
     maxHealth)  #init the Health
1355.                 self.intHealth=self.health
1356.                 self.regen= 4 if self.game.difficulty ==
     "Casual" else 2 #set regen amount
1357.                 self.regenTimer=25  #set regen timer
1358.                 self.regenCooldown=0  #set regen cooldown
1359.                 self.damageRed= 2 if self.game.difficulty ==
     "Casual" else 1  #give damage reduction in casual mode
1360.                 self.healthChange=False
1361.
1362.                 #init inventory
1363.                 self.inventory=inventory
1364.                 self.inventoryChange=True
1365.                 self.maxItemCount=32
1366.                 self.selectedIndex=0
1367.
1368.
1369.                 self.touching=None  #not touching a tile
1370.                 self.tool=None   #not having a tool
1371.                 self.event=None  #no tool event
1372.
1373.                 #initialise player images for animation
1374.
     self.playerSheet=Spritesheet("playersheet.png",3, (0,0,0), 16,
     25)
```

```
1375.
   self.walkingLeft=[self.playerSheet.getImage(i, 0) for i in
   range(13)]
1376.
   self.walkingRight=[self.playerSheet.getImage(i, 1) for i in
   range(13)]
1377.
   self.fallingLeft=self.playerSheet.getImage(13, 0)
1378.
   self.fallingRight=self.playerSheet.getImage(13, 1)
1379.                 self.usingLeft=[self.playerSheet.getImage(i,
   0) for i in range(14, 19)]
1380.                 self.usingRight=[self.playerSheet.getImage(i,
   1) for i in range(14, 19)]
1381.
1382.                 #initialise the image/rect and position
1383.                 self.x,self.y = x,y
1384.                 self.currentSprite=4
1385.
   self.image=self.walkingRight[self.currentSprite]
1386.                 self.rect = self.image.get_rect()
1387.                 self.rect.x, self.rect.y=x*TILESIZE,
   y*TILESIZE
1388.
1389.
1390.             #update the player
1391.
1392.             def update(self):
1393.                 self.updateInput()
1394.                 self.updateHealth()
1395.                 self.updateMotion()
1396.                 self.animate()
1397.
1398.             #make player move due to inputs
1399.             def updateInput(self):
1400.                 keys = pygame.key.get_pressed()
1401.                 if not(self.using):
1402.                     if keys[K_d]:
1403.                         self.vel.x = self.speed
1404.                         self.lookingLeft=False
1405.                         self.walking=True
1406.                     elif keys[K_a]:
1407.                         self.vel.x = -self.speed
1408.                         self.lookingLeft=True
1409.                         self.walking=True
1410.                     else:
1411.                         self.vel.x = 0
1412.                         self.walking=False
```

```
1413.
1414.                     if self.game.noclip:
1415.                         if keys[K_w]:
1416.                             self.vel.y=-self.speed
1417.                         elif keys[K_s]:
1418.                             self.vel.y=self.speed
1419.                         else:
1420.                             self.vel.y=0
1421.
1422.
1423.
1424.             #regen the health of the player if cooldowns are
    zero, decerment cooldowns and timers, kill if health is zero
    or lower
1425.             def updateHealth(self):
1426.                 if self.regenCooldown>0:
1427.                     self.regenCooldown-=1
1428.
1429.                 if self.regenTimer>0:
1430.                     self.regenTimer-=1
1431.
1432.                 if self.health<self.maxHealth and
    self.regenCooldown==0 and self.regenTimer==0:
1433.                     self.regenTimer=25
1434.
    self.heal(self.regen*max((self.health/self.maxHealth), 0.5))
1435.
1436.                 if self.health<=0:
1437.                     self.kill()
1438.
1439.             #deal damage if not in create mode and set a
    cooldown for the regen, flag health change for the HUD
1440.             def takeDamage(self, damage):
1441.                 if self.game.difficulty != "Creative":
1442.                     self.health-=
    round(damage/self.damageRed)
1443.                     self.healthChange=True
1444.                     self.regenCooldown=200
1445.
1446.                     self.updateHealth()
1447.
1448.             #heal an amount, flag health change for the HUD
1449.             def heal(self, amount):
1450.                 self.health += round(amount)
1451.
1452.                 if self.health> self.maxHealth:
1453.                     self.health=self.maxHealth
1454.
```

```
1455.                 self.healthChange=True
1456.
1457.                 self.updateHealth()
1458.
1459.
1460.             #update the motion and collisions
1461.             def updateMotion(self):
1462.                 self.fallVelocity=self.vel.y
1463.                 if not(self.game.noclip):
1464.                     self.vel.y += (self.grav * self.game.dt)
    #add gravity
1465.                     self.rect.y += round(self.vel.y *
    self.game.dt)  #change y
1466.                     self.grounded=False#make not grounded
    before checking the collsion
1467.                     self.collision("y")#collsion in y
1468.                     self.rect.x += round(self.vel.x *
    self.game.dt) #change x
1469.                     self.collision("x")#collision in x
1470.                 else:
1471.                     self.rect.y += round(self.vel.y *
    self.game.dt) #change y
1472.                     self.rect.x += round(self.vel.x *
    self.game.dt)#change x
1473.                 self.itemCollide()
1474.                 self.tileCollide()
1475.
1476.             #animate the player depending on what state they
    are in
1477.             def animate(self):
1478.                 if not(self.grounded):
1479.                     self.using=False
1480.                     self.event=False
1481.
1482.                     if self.tool:
1483.                         self.tool.kill()
1484.                         self.tool=None
1485.
1486.                     self.currentSprite=4
1487.
1488.                     if self.lookingLeft:
1489.                         self.image=self.fallingLeft
1490.                     else:
1491.                         self.image=self.fallingRight
1492.
1493.
1494.
1495.                 elif self.using:
```

```
1496.                     if self.inventory[self.selectedIndex][1]
   in self.game.toolImageDict:
1497.                         if not self.tool:
1498.                             self.tool=Tool(self.game,
   self.inventory[self.selectedIndex][1])
1499.
1500.
1501.                     self.currentSprite+=(self.tool.speed if
   self.tool else 0.4)*(60//self.game.FPS)
1502.                     if self.currentSprite >=
   len(self.usingRight):
1503.                         self.using=False
1504.                         self.finishUse()
1505.
1506.                         if self.tool:
1507.                             self.tool.kill()
1508.                             self.tool=None
1509.
1510.                         self.currentSprite=0
1511.
1512.                     if self.lookingLeft:
1513.                         self.image =
   self.usingLeft[int(self.currentSprite)]
1514.                         if self.tool:
1515.
   self.tool.updateFrame(int(self.currentSprite), True)
1516.
1517.                     else:
1518.                         self.image =
   self.usingRight[int(self.currentSprite)]
1519.                         if self.tool:
1520.
   self.tool.updateFrame(int(self.currentSprite), False)
1521.
1522.
1523.
1524.
1525.                 elif self.walking:
1526.
   self.currentSprite+=0.25*(60//self.game.FPS)
1527.
1528.                     if self.currentSprite >=
   len(self.walkingRight):
1529.                         self.currentSprite=0
1530.
1531.                     if self.lookingLeft:
1532.                         self.image =
   self.walkingLeft[int(self.currentSprite)]
```

```
1533.
1534.                     else:
1535.                         self.image =
  self.walkingRight[int(self.currentSprite)]
1536.
1537.                 else:
1538.                     self.currentSprite=4
1539.
1540.                     if self.lookingLeft:
1541.                         self.image =
  self.walkingLeft[int(self.currentSprite)]
1542.
1543.                     else:
1544.                         self.image =
  self.walkingRight[int(self.currentSprite)]
1545.
1546.
1547.
1548.
1549.
1550.             #if the player collides then make it so the
  player is moved to just outside of the intersection
1551.             def collision(self,dire):
1552.                 hits = pygame.sprite.spritecollide(self,
  self.game.colSprites, False,
  collided=pygame.sprite.collide_rect)
1553.
1554.                 if len(hits) == 0:
1555.                     return
1556.
1557.                 if dire == "y":
1558.                     for hit in hits:
1559.                         if self.vel.y > 0:
1560.                             if hit.rect.top <
  self.rect.bottom:
1561.
1562.                                 if self.fallVelocity>1000:
1563.
  self.takeDamage(self.fallVelocity/100)
1564.
1565.                                 self.vel.y = 0
1566.                                 self.rect.y = hit.rect.top -
  self.rect.height
1567.                                 self.grounded = True
1568.
1569.
1570.                         if self.vel.y < 0:
```

```
1571.                             if hit.rect.bottom >
   self.rect.top:
1572.                                 self.vel.y = 0
1573.                                 self.rect.y = hit.rect.bottom
1574.
1575.                 if dire == "x":
1576.                     for hit in hits:
1577.                         if self.vel.x > 0:
1578.                             if hit.rect.left <
   self.rect.right:
1579.                                 self.vel.x = 0
1580.                                 self.walking=False
1581.                                 self.rect.x = hit.rect.left -
   self.rect.width
1582.                         if self.vel.x < 0:
1583.                             if hit.rect.right >
   self.rect.left:
1584.                                 self.vel.x = 0
1585.                                 self.walking=False
1586.                                 self.rect.x = hit.rect.right
1587.
1588.             #check collisions with a crafting tile
1589.             def tileCollide(self):
1590.                 hits = pygame.sprite.spritecollide(self,
   self.game.tileSprites, False,
   collided=pygame.sprite.collide_rect)
1591.
1592.                 if len(hits) == 0:
1593.                     self.touching=None
1594.
1595.                 else:
1596.                     self.touching=hits[0].name
1597.
1598.             #check to pick up items
1599.             def itemCollide(self):
1600.                 hits = pygame.sprite.spritecollide(self,
   self.game.itemSprites, False,
   collided=pygame.sprite.collide_rect)
1601.
1602.                 if len(hits) == 0:
1603.                     return
1604.
1605.                 for hit in hits:
1606.                     self.itemPick(hit)
1607.
1608.         #     def itemPick(self, item):
1609.         #         found=False
1610.         #         for unit in self.inventory:
```

```
1611.       #             if unit[1]==item.name and unit[2]<
   self.maxItemCount and unit[0] < 40:
1612.       #
1613.       #
1614.       #                 self.inventory[unit[0]][2]+=1
1615.       #
1616.       #                 self.inventoryChange=True
1617.       #                 found=True
1618.       #                 item.kill()
1619.       #                 break
1620.       #
1621.       #         if not found:
1622.       #             for unit in self.inventory:
1623.       #                 if unit[2]==0 and unit[0] < 40:
1624.       #
   self.inventory[unit[0]][1]=item.name
1625.       #                     self.inventory[unit[0]][2]=1
1626.       #
1627.       #                     self.inventoryChange=True
1628.       #                     found=True
1629.       #                     item.kill()
1630.       #                     break
1631.
1632.           #pick the item
1633.           def itemPick(self, item):
1634.               item.kill()
1635.               self.addToInv(item.name, 1)
1636.           #allows for a change in the players inventory
1637.           def invChange(self, index, name, count):
1638.               # if the count is too large for the unit then
   refuse the change
1639.               if count > self.maxItemCount:
1640.                   return False
1641.
1642.               else:
1643.                   #if the count of the item is lower than
   zero set the unit to empty
1644.                   if count<1:
1645.
   self.inventory[index]=[index,"empty",0]
1646.
1647.                   else:
1648.
   self.inventory[index]=[index,name,count]
1649.
1650.                   self.inventoryChange=True
1651.
1652.                   return True
```

```
1653.
1654.             #scan the player inventory for a space and then
    add the item to that space, otherwise drop the item as an item
    object
1655.             def addToInv(self, name, count):
1656.                 found=False
1657.                 for unit in self.inventory:
1658.                     if unit[1]==name and unit[2] <
    self.maxItemCount and unit[0] < 40:
1659.
1660.                         count=unit[2]+count
1661.
1662.                         if count > self.maxItemCount:
1663.                             self.invChange(unit[0], name,
    self.maxItemCount)
1664.                             count-=self.maxItemCount
1665.                             self.addToInv(name,count)
1666.
1667.                         else:
1668.                             self.invChange(unit[0], name,
    count)
1669.                             count=0
1670.
1671.                         found=True
1672.                         break
1673.
1674.                 if not found:
1675.                     for unit in self.inventory:
1676.                         if unit[2]==0 and unit[0] < 40:
1677.
1678.                             if count > self.maxItemCount:
1679.                                 self.invChange(unit[0], name,
    self.maxItemCount)
1680.                                 count-=self.maxItemCount
1681.                                 self.addToInv(name,count)
1682.
1683.                             else:
1684.                                 self.invChange(unit[0], name,
    count)
1685.                                 count=0
1686.
1687.
1688.                             found=True
1689.                             break
1690.
1691.
1692.                 if not found:
1693.                     for i in range(count):
```

```
1694.
   Item(self.rect.x//TILESIZE,self.rect.y//TILESIZE-3,count)
1695.
1696.                     count=0
1697.
1698.             #remove the ingridients for crafting and then add
   the crafting result
1699.             def craft(self, item):
1700.                 tile=self.game.inv.tile
1701.
   ingNameList=list(self.game.inv.recipeDict[tile][item].keys())
1702.
   ingCountList=[self.game.inv.recipeDict[tile][item][ing] for
   ing in ingNameList]
1703.
1704.                 for index, ing in enumerate(ingNameList):
1705.                     for unit in self.inventory:
1706.                         if unit[1]==ing:
1707.                             ingCountList[index]-=unit[2]
1708.                             self.invChange(unit[0], unit[1],
   0)
1709.
1710.
1711.                     self.addToInv(ingNameList[index],
   int(ingCountList[index]*-1))
1712.
1713.                 self.addToInv(item,1)
1714.                 self.game.inv.updateCrafting()
1715.
1716.             #use an item
1717.             def use(self, event):
1718.                 self.vel.x=0
1719.                 if not self.using:
1720.                     self.using=True
1721.                     self.currentSprite=0
1722.
1723.                 self.event=event
1724.
1725.             #called from animiate if the animation is
   finished
1726.             def finishUse(self):
1727.                 #do an action
1728.                 if self.tool:
1729.                     if self.tool.type=="pickaxe":
1730.                         self.game.destroyBlock(self.event,
   self.tool)
1731.                     elif self.tool.type=="axe":
```

```
1732.                         self.game.destroyTree(self.event,
   self.tool)
1733.                     elif self.tool.type=="hammer":
1734.                         self.game.destroyWall(self.event,
   self.tool)
1735.                     elif self.tool.type=="hamaxe":
1736.                         self.game.destroyTree(self.event,
   self.tool)
1737.                         self.game.destroyWall(self.event,
   self.tool)
1738.
1739.                 else:
1740.
1741.                     self.game.place(self.event)
1742.
1743.                 self.event=None
1744.             #if the player is killed then he simply respawns
1745.             def kill(self):
1746.                 self.rect.x==20*TILESIZE
1747.                 self.rect.y=0*TILESIZE
1748.
1749.                 self.health=self.maxHealth//2
1750.
1751.                 for sprite in self.game.enemySprites:
1752.                     sprite.kill()
1753.
1754.         #enemy slime
1755.         class Slime(pygame.sprite.Sprite):
1756.             def __init__(self, game, x, y): # this x,y coord
   is a world coord (not pixels)
1757.                 self.game = game
1758.                 self.groups = (self.game.personSprites,
   self.game.enemySprites)
1759.                 pygame.sprite.Sprite.__init__(self,
   self.groups)
1760.
1761.                 self.vel = pygame.math.Vector2()
1762.
1763.                 #set constants similar to the player
1764.                 self.speed=50
1765.                 self.jumpSpeed=800
1766.                 self.grav = 2500    #effected by gravity
1767.                 self.grounded = True  #make it so it cannot
   double jump
1768.                 self.lookingLeft=True
1769.                 self.bouncing=False  #unique jumping
   animation effect
1770.
```

```
1771.                 self.jumpTimer=50  #cooldown for random jumps
1772.                 self.attackTimer=0 #cooldown for attacks
1773.                 self.stunTimer=0  #stun after getting hit
1774.
1775.                 self.maxHealth=40
1776.                 self.health=self.maxHealth
1777.
1778.                 #init the animation
1779.                 slimeSheet=Spritesheet("slimesheet.png",1,
   (0,0,0), 50, 25)
1780.                 self.bouncingList=[slimeSheet.getImage(i, 0)
   for i in range(17)]
1781.
1782.                 #init image and position
1783.                 self.x,self.y = x,y
1784.                 self.currentSprite=0
1785.
   self.image=self.bouncingList[self.currentSprite]
1786.
   self.mask=pygame.mask.from_surface(self.image)
1787.                 self.rect = self.image.get_rect()
1788.                 self.rect.x, self.rect.y=x*TILESIZE,
   y*TILESIZE
1789.
1790.
1791.             #call update for the slime
1792.             def update(self):
1793.                 self.AI()
1794.                 self.updateHealth()
1795.                 self.updateMotion()
1796.                 self.animate()
1797.                 self.playercollide()
1798.
1799.             #define the AI for the slime
1800.             def AI(self):
1801.                 #if the player enters within 250 pixels the
   slime will attack
1802.                 if self.game.calcDistanceSP(self,
   self.game.player) < 250:
1803.                     self.attack=True
1804.                 else:
1805.                     self.attack=False
1806.
1807.                 #if distance between slime and player exceed
   100 pixels then kill the sprite (optimisation purposes)
1808.                 if self.game.calcDistanceSP(self,
   self.game.player) > 1000:
1809.                     self.kill()
```

```
1810.
1811.                 #decrement stunTimer
1812.                 if self.stunTimer > 0:
1813.                     self.stunTimer-=1
1814.
1815.                 #if not stunned then do everything
1816.                 if self.stunTimer==0:
1817.                     #if the slime is in attack mode then make
    the space faster and jumps more frequent
1818.                     if self.attack:
1819.                         self.speed=100
1820.                         self.jumpTimer-=1
1821.
1822.                         #slime should face player while in
    attack mode
1823.                         if self.rect.x <
    self.game.player.rect.x:
1824.                             self.lookingLeft=False
1825.
1826.                         else:
1827.                             self.lookingLeft=True
1828.
1829.                     else:
1830.                         #normal speed/jump freq
1831.                         self.speed=50
1832.                         self.jumpTimer-=0.5
1833.
1834.                     #update the input fore the slime
1835.                     if self.lookingLeft:
1836.                         self.vel.x=-self.speed
1837.                     else:
1838.                         self.vel.x=self.speed
1839.
1840.                     #do small jumps every interval
1841.                     if self.jumpTimer==0:
1842.                         self.jump("small")
1843.                         self.jumpTimer=50
1844.
1845.                     #decrement attack timer
1846.                     if self.attackTimer!=0:
1847.                         self.attackTimer-=1
1848.
1849.             #jump fuction: takes size as an input and will
    decide what velocity jump to do based on this size and whether
    the slime is enraged
1850.             def jump(self, size):
1851.                 if self.grounded:
1852.                     if size=="large":
```

```
1853.                         if self.attack:
1854.                             vel=1000
1855.                         else:
1856.                             vel=500
1857.
1858.                     elif size=="small":
1859.                         if self.attack:
1860.                             vel=750
1861.                         else:
1862.                             vel=500
1863.
1864.                     else:
1865.                         vel=0
1866.
1867.                     self.vel.y=-vel
1868.                     self.bouncing=True
1869.
1870.             #set a transparency for the slime the lower the
    health goes
1871.             def updateHealth(self):
1872.
1873.
    self.image.set_alpha((self.health/self.maxHealth)*255)
1874.
1875.                 if self.health<=0:
1876.                     #kill slime if its health is zero or
    lower
1877.                     self.kill()
1878.
1879.             #make changes to the position of the slime and do
    collision in the x and y direction like the player
1880.             def updateMotion(self):
1881.                 self.fallVelocity=self.vel.y
1882.                 self.vel.y += (self.grav * self.game.dt)
1883.                 self.rect.y += round(self.vel.y *
    self.game.dt)
1884.                 self.grounded=False
1885.                 self.collision("y")
1886.                 self.rect.x += round(self.vel.x *
    self.game.dt)
1887.                 self.collision("x")
1888.
1889.             #animate the slime based on what mode it is in
1890.             def animate(self):
1891.                 #if jumping do the full animation
1892.                 if self.bouncing:
1893.
    self.currentSprite+=0.25*(60//self.game.FPS)
```

```
1894.
1895.                     if self.currentSprite >=
   len(self.bouncingList):
1896.                         self.currentSprite=0
1897.                         self.bouncing=False
1898.
1899.                 #otherwise play the first few frames
1900.                 else:
1901.
   self.currentSprite+=0.1*(60//self.game.FPS)
1902.                     if self.currentSprite >= 4:
1903.                         self.currentSprite=0
1904.
1905.                 #update the image and mask
1906.                 self.image =
   self.bouncingList[int(self.currentSprite)]
1907.
   self.mask=pygame.mask.from_surface(self.image)
1908.
1909.             #make the slime take damage and take knockback
   depending on the positional arugment from the player tool
1910.             def takeDamage(self,damage, pos=None):
1911.
1912.                 if pos:
1913.                     self.stunTimer=30
1914.
1915.                     self.jump("small")
1916.                     if pos == "right":
1917.                         self.vel.x+=(damage/self.health)*1000
1918.                     elif pos == "left":
1919.                         self.vel.x-=(damage/self.health)*1000
1920.
1921.                 self.health-=damage
1922.                 self.updateHealth()  #call to update the
   health
1923.
1924.
1925.
1926.
1927.             #checks for itersections betweem solid blocks and
   then places the sprite just before the intersection and sets
   the velocity to zero. This gives the impression of hitting a
   solid object.
1928.             #enemies do not take fall damage
1929.             def collision(self,dire):
1930.                 hits = pygame.sprite.spritecollide(self,
   self.game.colSprites, False,
   collided=pygame.sprite.collide_rect)
```

```
1931.                 if len(hits) == 0:
1932.                     return
1933.
1934.                 if dire == "y":
1935.                     for hit in hits:
1936.                         if self.vel.y > 0:
1937.                             if hit.rect.top <
   self.rect.bottom:
1938.
1939.
1940.                                 self.vel.y = 0
1941.                                 self.rect.y = hit.rect.top -
   self.rect.height
1942.                                 self.grounded = True
1943.
1944.
1945.                         if self.vel.y < 0:
1946.                             if hit.rect.bottom >
   self.rect.top:
1947.                                 self.vel.y = 0
1948.                                 self.rect.y = hit.rect.bottom
1949.
1950.                 if dire == "x":
1951.                     for hit in hits:
1952.                         if self.vel.x > 0:
1953.                             if hit.rect.left <
   self.rect.right:
1954.                                 self.vel.x = 0
1955.                                 self.rect.x = hit.rect.left -
   self.rect.width
1956.
1957.
1958.                         if self.vel.x < 0:
1959.                             if hit.rect.right >
   self.rect.left:
1960.                                 self.vel.x = 0
1961.                                 self.rect.x = hit.rect.right
1962.
1963.                     if not self.attack:
1964.                         self.lookingLeft =
   not(self.lookingLeft)
1965.                     else:
1966.                         self.jump("large")
1967.                                     # perform a large jump
   if the slime collides with a fall
1968.
1969.
```

```
            #if the slime hits the player then do damage if the attack timer is zero, if the damage is dealt then the attack timer is reset
            def playercollide(self):
                if self.attackTimer==0 and self.rect.colliderect(self.game.player.rect):
                    self.game.player.takeDamage(10)
                    self.attackTimer=80


        #zombie enemy
        class Zombie(pygame.sprite.Sprite):
            def __init__(self, game, x, y): # this x,y coord is a world coord (not pixels)
                self.game = game
                #initialise groups and the pygame.sprite inheritance
                self.groups = (self.game.personSprites, self.game.enemySprites)
                pygame.sprite.Sprite.__init__(self, self.groups)

                #set constants- effect from graviy, speed, health, animation sheet, current image
                self.vel = pygame.math.Vector2()

                self.speed=50
                self.grav = 2500
                self.grounded = True
                self.lookingLeft=True

                self.attackTimer=0
                self.stunTimer=0

                self.maxHealth=100
                self.health=self.maxHealth

                zombieSheet=Spritesheet("zombiesheet.png",1, (0,0,0), 50, 68)
                self.walkingRightList=[zombieSheet.getImage(i, 1) for i in range(3)] #need a seperate list for left and right as the right is in the flipped direction to the spritesheet
                self.walkingLeftList=[zombieSheet.getImage(i, 0) for i in range(3)]


                self.x,self.y = x,y
```

```
2005.                 self.currentSprite=0
2006.
   self.image=self.walkingRightList[self.currentSprite]
2007.
   self.mask=pygame.mask.from_surface(self.image)
2008.                 self.rect = self.image.get_rect()
2009.                 self.rect.x, self.rect.y=x*TILESIZE,
   y*TILESIZE
2010.
2011.             #call update
2012.             def update(self):
2013.                 self.AI()
2014.                 self.updateHealth()
2015.                 self.updateMotion()
2016.                 self.animate()
2017.                 self.playercollide()
2018.
2019.             #define AU
2020.             def AI(self):
2021.                 #if distance 300 pixels within player then
   attack
2022.                 if self.game.calcDistanceSP(self,
   self.game.player) < 300:
2023.                     self.attack=True
2024.                 else:
2025.                     self.attack=False
2026.
2027.
2028.                 if self.game.calcDistanceSP(self,
   self.game.player) > 1000:
2029.                     self.kill()
2030.
2031.
2032.                 if self.stunTimer > 0:
2033.                     self.stunTimer-=1
2034.
2035.                 #if attacking then face the player and run
   faster
2036.                 if self.stunTimer==0:
2037.                     if self.attack:
2038.                         self.speed=100
2039.
2040.                         if self.rect.x <
   self.game.player.rect.x:
2041.                             self.lookingLeft=False
2042.
2043.                         else:
2044.                             self.lookingLeft=True
```

```
2045.
2046.                   else:
2047.                       self.speed=50
2048.
2049.
2050.                   if self.lookingLeft:
2051.                       self.vel.x=-self.speed
2052.                   else:
2053.                       self.vel.x=self.speed
2054.
2055.
2056.                   if self.attackTimer!=0:
2057.                       self.attackTimer-=1
2058.
2059.           #jump fuction same as slime
2060.           def jump(self, size):
2061.               if self.grounded:
2062.                   if size=="large":
2063.                       if self.attack:
2064.                           vel=750
2065.                       else:
2066.                           vel=500
2067.
2068.                   elif size=="small":
2069.                       if self.attack:
2070.                           vel=500
2071.                       else:
2072.                           vel=500
2073.
2074.                   else:
2075.                       vel=0
2076.
2077.                   self.vel.y=-vel
2078.                   self.bouncing=True
2079.
2080.           #update health is the same but on kill the zombie
   has a chance to drop rare ores
2081.           def updateHealth(self):
2082.
2083.
   self.image.set_alpha((self.health/self.maxHealth)*255)
2084.
2085.               if self.health<=0:
2086.                   self.kill()
2087.                   if random.uniform<=0.03:
2088.                       Item(self.rect.x,self.rect.y,"uru")
2089.                   elif random.uniform<=0.05:
```

```
2090.
   Item(self.rect.x,self.rect.y,"adamantite")
2091.
2092.           #updates positon of zombie and calls for
   collisions to be dealt with
2093.           def updateMotion(self):
2094.               self.fallVelocity=self.vel.y
2095.               self.vel.y += (self.grav * self.game.dt)
2096.               self.rect.y += round(self.vel.y *
   self.game.dt)
2097.               self.grounded=False
2098.               self.collision("y")
2099.               self.rect.x += round(self.vel.x *
   self.game.dt)
2100.               self.collision("x")
2101.
2102.           #animate the zombie
2103.           def animate(self):
2104.               self.currentSprite+=0.1*(60//self.game.FPS)
2105.
2106.               if self.currentSprite >=
   len(self.walkingRightList):
2107.                   self.currentSprite=0
2108.
2109.               if self.lookingLeft:
2110.                   self.image =
   self.walkingLeftList[int(self.currentSprite)]
2111.               else:
2112.                   self.image =
   self.walkingRightList[int(self.currentSprite)]
2113.
2114.
2115.
   self.mask=pygame.mask.from_surface(self.image)
2116.
2117.            # take damage and be knockbacked
2118.           def takeDamage(self,damage, pos=None):
2119.
2120.               if pos:
2121.                   self.stunTimer=30
2122.
2123.                   self.jump("small")
2124.                   if pos == "right":
2125.                       self.vel.x+=(damage/self.health)*1000
2126.                   elif pos == "left":
2127.                       self.vel.x-=(damage/self.health)*1000
2128.
2129.               self.health-=damage
```

```
2130.                 self.updateHealth()
2131.
2132.
2133.
2134.             #collision is the same as player
2135.
2136.             def collision(self,dire):
2137.                 hits = pygame.sprite.spritecollide(self,
   self.game.colSprites, False,
   collided=pygame.sprite.collide_rect)
2138.                 if len(hits) == 0:
2139.                     return
2140.
2141.                 if dire == "y":
2142.                     for hit in hits:
2143.                         if self.vel.y > 0:
2144.                             if hit.rect.top <
   self.rect.bottom:
2145.
2146.         #                         if self.fallVelocity>1500:
2147.         #                             self.takeDamage(50)
2148.
2149.                                 self.vel.y = 0
2150.                                 self.rect.y = hit.rect.top -
   self.rect.height
2151.                                 self.grounded = True
2152.
2153.
2154.                         if self.vel.y < 0:
2155.                             if hit.rect.bottom >
   self.rect.top:
2156.                                 self.vel.y = 0
2157.                                 self.rect.y = hit.rect.bottom
2158.
2159.                                 self.lookingLeft =
   not(self.lookingLeft)
2160.
2161.                 if dire == "x":
2162.                     for hit in hits:
2163.                         if self.vel.x > 0:
2164.                             if hit.rect.left <
   self.rect.right:
2165.                                 self.vel.x = 0
2166.                                 self.rect.x = hit.rect.left -
   self.rect.width
2167.
2168.                         if self.vel.x < 0:
```

```
2169.                             if hit.rect.right >
   self.rect.left:
2170.                                 self.vel.x = 0
2171.                                 self.rect.x = hit.rect.right
2172.
2173.
2174.                         if not self.attack:
2175.                             self.jump("small")  #zombie will
   do a jump on wall collide
2176.                         else:
2177.                             self.jump("large")
2178.
2179.             #damage works the same as slime
2180.             def playercollide(self):
2181.                 if self.attackTimer==0 and
   self.rect.colliderect(self.game.player.rect):
2182.                     self.game.player.takeDamage(30)
2183.                     self.attackTimer=120
2184.
2185.
2186.         #everything works the same as zombie just a fiew
   changes
2187.         class Skeleton(pygame.sprite.Sprite):
2188.             def __init__(self, game, x, y): # this x,y coord
   is a world coord (not pixels)
2189.                 self.game = game
2190.                 self.groups = (self.game.personSprites,
   self.game.enemySprites)
2191.                 pygame.sprite.Sprite.__init__(self,
   self.groups)
2192.
2193.                 self.vel = pygame.math.Vector2()
2194.
2195.                 self.speed=50
2196.                 self.jumpSpeed=800
2197.                 self.grav = 2500
2198.                 self.grounded = True
2199.                 self.lookingLeft=True
2200.                 self.using=False
2201.
2202.                 self.attackTimer=0
2203.                 self.stunTimer=0
2204.
2205.                 self.maxHealth=200  #higher health than
   zombie
2206.                 self.health=self.maxHealth
2207.
```

```
2208.
   skeletonSheet=Spritesheet("skeletonsheet.png",3, (0,0,0), 16,
   25)
2209.                 self.walkingRight=[skeletonSheet.getImage(i,
   1) for i in range(10)]
2210.                 self.walkingLeft=[skeletonSheet.getImage(i,
   0) for i in range(10)]
2211.                 self.usingRight=[skeletonSheet.getImage(i, 1)
   for i in range(10,16)]
2212.                 self.usingLeft=[skeletonSheet.getImage(i, 0)
   for i in range(10,16)]
2213.
2214.                 self.x,self.y = x,y
2215.                 self.currentSprite=0
2216.
   self.image=self.walkingRight[self.currentSprite]
2217.
   self.mask=pygame.mask.from_surface(self.image)
2218.                 self.rect = self.image.get_rect()
2219.                 self.rect.x, self.rect.y=x*TILESIZE,
   y*TILESIZE
2220.
2221.
2222.             def update(self):
2223.                 self.AI()
2224.                 self.updateHealth()
2225.                 self.updateMotion()
2226.                 self.playercollide()
2227.                 self.animate()
2228.
2229.
2230.             def AI(self):
2231.                 if self.game.calcDistanceSP(self,
   self.game.player) < 300:
2232.                     self.attack=True
2233.                 else:
2234.                     self.attack=False
2235.
2236.                 if self.game.calcDistanceSP(self,
   self.game.player) > 1000:
2237.                     self.kill()
2238.
2239.                 if self.stunTimer > 0:
2240.                     self.stunTimer-=1
2241.
2242.                 if self.stunTimer==0:
2243.                     if self.attack:
2244.                         self.speed=100
```

```
2245.
2246.                         if self.rect.x <
   self.game.player.rect.x:
2247.                             self.lookingLeft=False
2248.
2249.
2250.                         else:
2251.                             self.lookingLeft=True
2252.
2253.                         if self.attackTimer==0:
2254.                             self.use()    #has the ability to
   shoot projectile
2255.
2256.                     else:
2257.                         self.speed=50
2258.
2259.
2260.                     if not self.using:
2261.                         if self.lookingLeft:
2262.                             self.vel.x=-self.speed
2263.                         else:
2264.                             self.vel.x=self.speed
2265.
2266.
2267.                         if self.attackTimer!=0:
2268.                             self.attackTimer-=1
2269.
2270.
2271.             def jump(self, size):
2272.                 if size=="large":
2273.                     if self.attack:
2274.                         vel=750
2275.                     else:
2276.                         vel=500
2277.
2278.                 elif size=="small":
2279.                     if self.attack:
2280.                         vel=500
2281.                     else:
2282.                         vel=500
2283.
2284.                 else:
2285.                     vel=0
2286.
2287.                 self.vel.y=-vel
2288.
2289.
2290.             def updateHealth(self):
```

```
2291.
2292.
    self.image.set_alpha((self.health/self.maxHealth)*255)
2293.
2294.                 if self.health<=0:
2295.                     self.kill()
2296.                     if random.uniform<=0.06:
2297.                         Item(self.rect.x,self.rect.y,"uru")
2298.                     elif random.uniform<=0.10:
2299.
    Item(self.rect.x,self.rect.y,"adamantite")
2300.
2301.             def updateMotion(self):
2302.                 self.fallVelocity=self.vel.y
2303.                 self.vel.y += (self.grav * self.game.dt)
2304.                 self.rect.y += round(self.vel.y *
    self.game.dt)
2305.                 self.grounded=False
2306.                 self.collision("y")
2307.                 self.rect.x += round(self.vel.x *
    self.game.dt)
2308.                 self.collision("x")
2309.
2310.
2311.             def animate(self):
2312.
2313.                 if self.using:
2314.
2315.
    self.currentSprite+=0.25*(60//self.game.FPS)
2316.                     if self.currentSprite >=
    len(self.usingRight):
2317.                         self.using=False
2318.                         self.finishUse()
2319.                         self.currentSprite=0
2320.
2321.                     if self.lookingLeft:
2322.                         self.image =
    self.usingLeft[int(self.currentSprite)]
2323.
2324.
2325.                     else:
2326.                         self.image =
    self.usingRight[int(self.currentSprite)]
2327.
2328.
2329.                 else:
```

```
2330.
   self.currentSprite+=0.1*(60//self.game.FPS)
2331.
2332.                   if self.currentSprite >=
   len(self.walkingRight):
2333.                       self.currentSprite=0
2334.
2335.                   if self.lookingLeft:
2336.                       self.image =
   self.walkingLeft[int(self.currentSprite)]
2337.
2338.                   else:
2339.                       self.image =
   self.walkingRight[int(self.currentSprite)]
2340.
2341.
2342.
   self.mask=pygame.mask.from_surface(self.image)
2343.
2344.
2345.           def takeDamage(self,damage, pos=None):
2346.
2347.               if pos:
2348.                   self.stunTimer=30
2349.
2350.                   self.jump("small")
2351.                   if pos == "right":
2352.                       self.vel.x+=(damage/self.health)*1000
2353.                   elif pos == "left":
2354.                       self.vel.x-=(damage/self.health)*1000
2355.
2356.               self.health-=damage
2357.               self.updateHealth()
2358.
2359.
2360.
2361.
2362.           def collision(self,dire):
2363.               hits = pygame.sprite.spritecollide(self,
   self.game.colSprites, False,
   collided=pygame.sprite.collide_rect)
2364.               if len(hits) == 0:
2365.                   return
2366.
2367.               if dire == "y":
2368.                   for hit in hits:
2369.                       if self.vel.y > 0:
```

```
2370.                             if hit.rect.top <
   self.rect.bottom:
2371.
2372.         #                       if self.fallVelocity>1500:
2373.         #                           self.takeDamage(50)
2374.
2375.                                 self.vel.y = 0
2376.                                 self.rect.y = hit.rect.top -
   self.rect.height
2377.                                 self.grounded = True
2378.
2379.
2380.                         if self.vel.y < 0:
2381.                             if hit.rect.bottom >
   self.rect.top:
2382.                                 self.vel.y = 0
2383.                                 self.rect.y = hit.rect.bottom
2384.
2385.                                 self.lookingLeft =
   not(self.lookingLeft)
2386.
2387.                 if dire == "x":
2388.                     for hit in hits:
2389.                         if self.vel.x > 0:
2390.                             if hit.rect.left <
   self.rect.right:
2391.                                 self.vel.x = 0
2392.                                 self.rect.x = hit.rect.left -
   self.rect.width
2393.
2394.                         if self.vel.x < 0:
2395.                             if hit.rect.right >
   self.rect.left:
2396.                                 self.vel.x = 0
2397.                                 self.rect.x = hit.rect.right
2398.
2399.
2400.                         if not self.attack:
2401.                             self.jump("small")
2402.                         else:
2403.                             self.jump("large")
2404.
2405.
2406.
2407.             def playercollide(self):
2408.                 if self.attackTimer==0 and
   self.rect.colliderect(self.game.player.rect):
2409.                     self.game.player.takeDamage(70)
```

```
2410.                     self.attackTimer=120
2411.
2412.
2413.             #skeleton can use fireball
2414.             def use(self):
2415.                 self.vel.x=0
2416.                 if not self.using and not
   self.rect.colliderect(self.game.player.rect):
2417.                     self.using=True
2418.                     self.currentSprite=0
2419.                     self.attackTimer=120
2420.
2421.             #calculates the angle using pygame.math.vector
2422.             def finishUse(self):
2423.                 playerPos = pygame.math.Vector2()
2424.                 playerPos[0] = self.game.player.rect.x
2425.                 playerPos[1] = self.game.player.rect.y
2426.                 skeletonPos= pygame.math.Vector2()
2427.                 skeletonPos[0] = self.rect.x
2428.                 skeletonPos[1] = self.rect.y
2429.                 angle=skeletonPos.angle_to(playerPos)
2430.                 self.shoot(angle)
2431.
2432.             #shoots fireball at this angle by seperating the
   velocity into components
2433.             def shoot(self, angle):
2434.                 vel=500
2435.                 velX=-1*vel*cos(angle*(pi/180)) if
   self.lookingLeft else vel*cos(angle*(pi/180))
2436.                 velY=vel*sin(angle*(pi/180))
2437.                 Fireball(self.game, self.rect.x, self.rect.y,
   velX, velY, False, 30)
2438.
2439.
2440.         #projectile class
2441.         class Fireball(pygame.sprite.Sprite):
2442.             def __init__(self,game,x,y, velX, velY, friendly,
   damage):
2443.                 self.game = game
2444.
2445.                 self.vel = pygame.math.Vector2()
2446.
2447.                 self.groups = (self.game.camSprites,
   self.game.enemySprites)
2448.
   pygame.sprite.Sprite.__init__(self,self.groups)
2449.
```

```
2450.                 sheet=Spritesheet("pinkfireball.png",2,
   (0,0,0), 9,9)
2451.                 self.animationList=[sheet.getImage(i, 0) for
   i in range(12)]
2452.
2453.                 self.damage=damage
2454.                 self.time=100
2455.                 self.friendly= friendly
2456.
2457.                 self.currentSprite=0
2458.
   self.image=self.animationList[self.currentSprite]
2459.                 self.size=self.image.get_size()
2460.                 self.rect = self.image.get_rect()
2461.                 self.rect.x,self.rect.y = x,y
2462.
2463.                 self.vel.x=velX
2464.                 self.vel.y=velY
2465.
2466.             #will kill after a certain time
2467.             def update(self):
2468.                 self.time-=1
2469.                 if self.time==0:
2470.                     self.kill()
2471.
2472.                 self.updateMotion()
2473.                 self.collide()
2474.                 self.animate()
2475.
2476.             #goes stright along a single line
2477.             def updateMotion(self):
2478.                 self.collision()
2479.                 self.rect.y += round(self.vel.y *
   self.game.dt)
2480.                 self.rect.x += round(self.vel.x *
   self.game.dt)
2481.
2482.
2483.             #will look as if sparkle through the animation
2484.             def animate(self):
2485.                 self.currentSprite+=0.1*(60//self.game.FPS)
2486.
2487.                 if self.currentSprite >=
   len(self.animationList):
2488.                     self.currentSprite=0
2489.
2490.                 self.image =
   self.animationList[int(self.currentSprite)]
```

```
2491.
   self.mask=pygame.mask.from_surface(self.image)
2492.
2493.            #dissipates if it hits something that didn't cast
   it
2494.            def takeDamage(self,damage, pos=None):
2495.                if not self.friendly:
2496.                    self.kill()
2497.
2498.            #on colllision with solid block it dissipates
2499.            def collision(self):
2500.                hits = pygame.sprite.spritecollide(self,
   self.game.colSprites, False,
   collided=pygame.sprite.collide_rect)
2501.
2502.                if len(hits)>0:
2503.                    self.kill()
2504.
2505.            #if it collides with personSprite that didn't
   fire it then it will do damage
2506.            def collide(self):
2507.                hits = pygame.sprite.spritecollide(self,
   self.game.personSprites, False)
2508.                if len(hits)==0:
2509.                    return
2510.
2511.                else:
2512.                    hit=hits[0]
2513.                    if hit.__class__.__name__=="Player":
2514.                        if not self.friendly:
2515.                            hit.takeDamage(self.damage)
2516.                            self.kill()
2517.
2518.                    elif hit.__class__.__name__=="Fireball":
2519.                        self.kill()
2520.
2521.                    else:
2522.                        if self.friendly:
2523.                            hit.takeDamage(self.damage)
2524.                            self.kill()
2525.
2526.
2527.
2528.
2529.
2530.        #class for a solid block
2531.        class Block(pygame.sprite.Sprite):
2532.            def __init__(self,game,x,y, name):
```

```
2533.                 self.game = game
2534.                 self.name=name
2535.                 #init groups
2536.                 self.groups = (self.game.camSprites,
   self.game.colSprites)
2537.
   pygame.sprite.Sprite.__init__(self,self.groups)
2538.                 #add to the dictionary for block/tile objects
   --> follow darkness rules
2539.                 self.x,self.y = x,y
2540.                 self.game.gridDict[str((self.x,self.y))]=self
2541.                 self.darkness=0
2542.
2543.                 #pulls the health from the stat dictionary
2544.                 self.maxHealth =
   self.game.blockStatDict[self.name]["health"]
2545.                 self.health=self.maxHealth
2546.
2547.                 self.image =
   pygame.Surface((TILESIZE,TILESIZE)).convert_alpha()
2548.
   self.image.blit(self.game.blockImageDict[self.name], (0,0))
2549.                 self.size=self.image.get_size()
2550.                 self.rect = self.image.get_rect()
2551.                 self.rect.x,self.rect.y =
   self.x*TILESIZE,self.y*TILESIZE
2552.
2553.             #deal damage to the block to increase
   transparency, damage can only be dealt if the block can be
   destroyed in less than 5 hits, otherwise no damage dealt.
2554.             #once health is zero the block is killed and an
   item is dropped
2555.                 #destroyed with pickaxe
2556.
2557.             def damage(self, power):
2558.                 if self.maxHealth//power <=5 and
   self.darkness==0:
2559.                     self.health-=power
2560.
   self.image.set_alpha(int((self.health/self.maxHealth)*255))
2561.
2562.                 if self.health <=0:
2563.                     del
   self.game.gridDict[str((self.x,self.y))]
2564.                     self.game.updateDarknessAR(self)
2565.
2566.                     if str((self.x,self.y)) in
   self.game.wallDict:
```

```
2567.
   self.game.wallDict[str((self.x,self.y))].add(self.game.backSpr
   ites,self.game.wallSprites)
2568.                   elif str((self.x,self.y)) in
   self.game.backDict:
2569.
   self.game.backDict[str((self.x,self.y))].add(self.game.backSpr
   ites,self.game.backgroundSprites)
2570.
2571.                   if self.name=="grass":
2572.
   Item(self.game,self.rect.x,self.rect.y,"dirt")
2573.
2574.                   else:
2575.
   Item(self.game,self.rect.x,self.rect.y,self.name)
2576.
2577.                   self.kill()
2578.
2579.           #update the darkness level of the block
2580.           def updateDarkness(self,level):
2581.               self.darkness=level
2582.               self.image =
   pygame.Surface((TILESIZE,TILESIZE)).convert_alpha()
2583.
2584.               if self.darkness < 3:
2585.
   self.image.blit(self.game.blockImageDict[self.name], (0,0))
2586.
   self.dark=pygame.Surface((TILESIZE,TILESIZE)).convert_alpha()
2587.
   self.dark.set_alpha(int((255/3)*self.darkness))
2588.                   self.image.blit(self.dark, (0,0))
2589.
2590.           def __str__(self):
2591.               return("self.x,self.y,self.name")
2592.
2593.
2594.       #wall, same as a block but you can walk through it
   and doesn't have a darkness value to it
2595.       class Wall(pygame.sprite.Sprite):
2596.           def __init__(self,game,x,y, name):
2597.               self.game = game
2598.               self.name=name
2599.
2600.               self.groups = (self.game.backSprites,
   self.game.wallSprites)
```

```
2601.
   pygame.sprite.Sprite.__init__(self,self.groups)
2602.
2603.                 self.x,self.y = x,y
2604.                 self.game.wallDict[str((self.x,self.y))]=self
2605.
2606.                 self.maxHealth = 30
2607.                 self.health=self.maxHealth
2608.
2609.                 self.image =
   pygame.Surface((TILESIZE,TILESIZE)).convert_alpha()
2610.
   self.image.blit(self.game.wallImageDict[self.name], (0,0))
2611.                 self.size=self.image.get_size()
2612.                 self.rect = self.image.get_rect()
2613.                 self.rect.x,self.rect.y =
   self.x*TILESIZE,self.y*TILESIZE
2614.
2615.             #destroyed with hamemr
2616.             def damage(self, power):
2617.                 self.health-=power
2618.
   self.image.set_alpha(int((self.health/self.maxHealth)*255))
2619.
2620.                 if self.health <=0:
2621.                     if str((self.x,self.y)) in
   self.game.backDict:
2622.
   self.game.backDict[str((self.x,self.y))].add(self.game.camSpri
   tes,self.game.backSprites)
2623.
2624.                     self.kill()
2625.
2626.             def __str__(self):
2627.                 return("self.x,self.y,self.name")
2628.
2629.
2630.         #same as block but is not collideable
2631.         class Tile(pygame.sprite.Sprite):
2632.             def __init__(self,game,x,y, name):
2633.                 self.game = game
2634.                 self.name=name
2635.
2636.                 self.groups = (self.game.camSprites,
   self.game.tileSprites, self.game.wallSprites)
2637.
   pygame.sprite.Sprite.__init__(self,self.groups)
2638.
```

```
2639.
2640.                 self.x,self.y = x,y
2641.                 self.game.gridDict[str((self.x,self.y))]=self
2642.                 self.darkness=0
2643.
2644.                 self.maxHealth = 30
2645.                 self.health=self.maxHealth
2646.
2647.                 self.image =
   pygame.Surface((TILESIZE*2,TILESIZE*2)).convert_alpha()
2648.
   self.image.blit(self.game.tileImageDict[self.name], (0,0))
2649.                 self.image.set_colorkey((0,0,0))
2650.                 self.size=self.image.get_size()
2651.                 self.rect = self.image.get_rect()
2652.                 self.rect.x,self.rect.y =
   self.x*TILESIZE,self.y*TILESIZE
2653.
2654.                 if pygame.sprite.spritecollideany(self,
   self.game.colSprites):
2655.                     self.damage(30)
2656.
2657.             #destroy with hammer
2658.             def damage(self, power):
2659.                 self.health-=power
2660.
   self.image.set_alpha(int((self.health/self.maxHealth)*255))
2661.
2662.                 if self.health <=0:
2663.
2664.
   Item(self.game,self.rect.x,self.rect.y,self.name)
2665.
2666.                     self.kill()
2667.
2668.             def updateDarkness(self,level):
2669.                 self.image =
   pygame.Surface((TILESIZE*2,TILESIZE*2)).convert_alpha()
2670.                 self.image.set_colorkey((0,0,0))
2671.
   self.image.blit(self.game.tileImageDict[self.name], (0,0))
2672.                 self.darkness=level
2673.
   self.dark=pygame.Surface((TILESIZE,TILESIZE)).convert_alpha()
2674.
   self.dark.set_alpha(int((255/3.1)*self.darkness))
2675.                 self.image.blit(self.dark, (0,0))
2676.
```

```
2677.           def __str__(self):
2678.               return("self.x,self.y,self.name")
2679.
2680.       #destroy with axe to get wood
2681.       class Tree(pygame.sprite.Sprite):
2682.           def __init__(self,game,x,y):
2683.               self.game = game
2684.
2685.               self.groups = (self.game.camSprites,
   self.game.treeSprites)
2686.
   pygame.sprite.Sprite.__init__(self,self.groups)
2687.
2688.               self.maxHealth = 100
2689.               self.health=self.maxHealth
2690.
2691.               self.x=x
2692.               self.y=y
2693.
2694.               self.image =
   pygame.image.load(os.path.expanduser(filepathG +
   "tree.png")).convert_alpha()
2695.               self.size=self.image.get_size()
2696.               self.rect = self.image.get_rect()
2697.
2698.               self.rect.x=x
2699.               self.rect.bottom=y
2700.
   self.mask=pygame.mask.from_surface(self.image)
2701.
2702.           def damage(self, power):
2703.
2704.               self.health-=power
2705.
   self.image.set_alpha(int((self.health/self.maxHealth)*255))
2706.
2707.               if self.health <=0:
2708.
2709.                   for i in range(10):
2710.
   Item(self.game,self.rect.x+35,self.rect.y+160,"wood")
2711.
2712.
2713.                   self.kill()
2714.
2715.
2716.           def __str__(self):
2717.               return("self.x,self.y")
```

```
2718.
2719.       #pick up to get the item
2720.       class Item(pygame.sprite.Sprite):
2721.               def __init__(self,game,x,y, name):
2722.                   self.game = game
2723.                   self.name = name
2724.                   self.scale=0.5
2725.
2726.                   self.groups = (self.game.camSprites,
   self.game.itemSprites)
2727.
   pygame.sprite.Sprite.__init__(self,self.groups)
2728.
2729.                   self.image =
   (self.game.itemImageDict[self.name]).convert_alpha()
2730.                   self.size=self.image.get_size()
2731.                   self.image =
   pygame.transform.scale(self.image, (int(self.size[0] *
   self.scale), int(self.size[1] * self.scale))).convert_alpha()
2732.                   self.rect = self.image.get_rect()
2733.                   self.rect.x,self.rect.y = x+10,y+10
2734.
2735.
2736.               def __str__(self):
2737.                   return f"{self.name} item at
   ({self.x},{self.y}) at "
2738.
2739.       #acts as a wall but is unbreakable
2740.       class Background(pygame.sprite.Sprite):
2741.           def __init__(self,game,x,y, name):
2742.               self.game = game
2743.               self.name = name
2744.
2745.
2746.               self.groups = (self.game.backSprites)
2747.
   pygame.sprite.Sprite.__init__(self,self.groups)
2748.
2749.               self.x,self.y = x,y
2750.               self.game.backDict[str((self.x,self.y))]=self
2751.
2752.               self.image =
   pygame.Surface((TILESIZE,TILESIZE)).convert_alpha()
2753.
   self.image.blit(self.game.backImageDict[self.name], (0,0))
2754.               self.size=self.image.get_size()
2755.               self.rect = self.image.get_rect()
```

```
2756.                 self.rect.x,self.rect.y =
   self.x*TILESIZE,self.y*TILESIZE
2757.
2758.             def __str__(self):
2759.                 return("self.x,self.y,self.name")
2760.
2761.         #group for sprites that stay in the same place
   (relative)
2762.         class CameraGroup(pygame.sprite.Group):
2763.
2764.             def __init__(self,game):
2765.                 pygame.sprite.Group.__init__(self)
2766.                 self.game = game
2767.                 self.halfWidth =
   self.game.screen.get_size()[0]//2  # so player always in
   middle
2768.                 self.halfHeight =
   self.game.screen.get_size()[1] // 2
2769.                 self.offset = pygame.math.Vector2() # vector
   currPos to middle
2770.
2771.
2772.             def customDraw(self):
2773.                 self.offset.x = self.halfWidth -
   self.game.player.rect.centerx
2774.                 self.offset.y = self.halfHeight -
   self.game.player.rect.centery
2775.                 self.game.off = self.offset
2776.                 for sprite in self.sprites(): # overlapping
   should depend on if under or top
2777.                     newPos = sprite.rect.topleft +
   self.offset
2778.                     if -125<newPos[0]<1100 and -
   200<newPos[1]<800:
2779.
   self.game.screen.blit(sprite.image,newPos)
2780.
2781.         #group for sprites that move with the player
   (relative)
2782.         class HudGroup(pygame.sprite.Group):
2783.
2784.             def __init__(self,game):
2785.                 pygame.sprite.Group.__init__(self)
2786.                 self.game = game
2787.
2788.             def customDraw(self):
2789.                 for sprite in self.sprites():
```

```
2790.
   self.game.screen.blit(sprite.image,(sprite.rect.x,sprite.rect.
   y))
2791.
2792.
2793.        g = Game(screen, worldName, difficulty, prevTime)
2794.        g.main()
2795.
2796.    #menu fuction
2797.    def menu(screen):
2798.        #menu option templates
2799.        class Entry(pygame.sprite.Sprite):
2800.            def __init__(self):
2801.                self.BaseColour=((58,73,135))
2802.                self.SelectedColour=((235,235,85))
2803.                self.chosen=False
2804.                self.chosenColour=((255,0,0))
2805.
2806.                self.image=pygame.Surface((600,75))
2807.                self.image.set_alpha(200)
2808.                self.image.fill(self.BaseColour)
2809.
2810.                self.rect = self.image.get_rect()
2811.                self.rect.x = 50
2812.                self.rect.y = 175
2813.
2814.        class Button(pygame.sprite.Sprite):
2815.            def __init__(self, image, x, y):
2816.
   self.imageBase=pygame.image.load(os.path.expanduser(filepathM
   + f"{image}.png"))
2817.
   self.imageSelected=pygame.image.load(os.path.expanduser(filepa
   thM + f"{image}Selected.png"))
2818.                self.image=self.imageBase
2819.
2820.                self.rect = self.image.get_rect()
2821.                self.rect.x=x
2822.                self.rect.y=y
2823.
2824.        class Image(pygame.sprite.Sprite):
2825.            def __init__(self, image, x, y):
2826.
   self.image=pygame.image.load(os.path.expanduser(filepathM +
   f"{image}.png"))
2827.
2828.                self.rect = self.image.get_rect()
2829.                self.rect.x=x
```

```
2830.                 self.rect.y=y
2831.
2832.         class Box(pygame.sprite.Sprite):
2833.             def __init__(self, surface, fill, alpha, x, y):
2834.                 self.image=pygame.Surface(surface)
2835.                 self.image.fill(fill)
2836.                 self.image.set_alpha(alpha)
2837.
2838.                 self.rect = self.image.get_rect()
2839.                 self.rect.x = x
2840.                 self.rect.y = y
2841.
2842.         class Text(pygame.sprite.Sprite):
2843.             def __init__(self, text, size, colour, timer, x,
    y):
2844.                 self.text=text
2845.                 self.fontSize=size
2846.                 self.font="AndyBold"
2847.                 self.colour=colour
2848.                 self.timer=timer
2849.                 self.x=x
2850.                 self.y=y
2851.
2852.         #menu subroutines
2853.         def checkSelected(obj):
2854.             pos = pygame.mouse.get_pos()
2855.
2856.             if obj.rect.collidepoint(pos):
2857.
2858.                 if obj.__class__.__base__.__name__ ==
    "Entry":
2859.
2860.                     if obj.chosen:
2861.                         obj.image.fill(obj.chosenColour)
2862.                         textToReturn=obj.text
2863.                     else:
2864.                         obj.image.fill(obj.SelectedColour)
2865.
2866.                 else:
2867.                     obj.image=obj.imageSelected
2868.
2869.                 return True
2870.
2871.             else:
2872.                 if obj.__class__.__base__.__name__ ==
    "Entry":
2873.
2874.                     if obj.chosen:
```

```
                        obj.image.fill(obj.chosenColour)
                        textToReturn=obj.text
                    else:
                        obj.image.fill(obj.BaseColour)

                else:
                    obj.image=obj.imageBase

                return False


        def drawList(objList, screen):
            for obj in objList:
                if obj.__class__.__name__ == "Text":
                    font=pygame.font.SysFont(obj.font,
    obj.fontSize)
                    text=font.render(obj.text, 1,
    self.colour)
                    screen.blit(text, (obj.x, obj.y))

                else:
                    screen.blit(obj.image,
    (obj.rect.x,obj.rect.y))

                    if obj.__class__.__base__.__name__ ==
    "Entry" or obj.__class__.__name__ == "Button":
                        checkSelected(obj)

                        if obj.__class__.__base__.__name__ ==
    "Entry":

                            if type(obj).__name__ ==
    "NameEntry":
                                obj.text=f"Name:
    [{str(obj.name)}] Difficulty: [{str(obj.difficulty)}]"


    font=pygame.font.SysFont("AndyBold", 28)
                            text=font.render(obj.text, 1,
    (255,255,255))
                            screen.blit(text, (obj.rect.x+5,
    obj.rect.y+10))

        def displayText(obj):
            font=pygame.font.SysFont(obj.font, obj.fontSize)
            text=font.render(obj.text, 1, obj.colour)
            screen.blit(text, (obj.x, obj.y))

```

```
2913.
2914.       def update(B1,B2):
2915.           B1.rect.x-=1
2916.           B2.rect.x-=1
2917.           if B1.rect.x==-1000:
2918.               B1.rect.x=1000
2919.           elif B2.rect.x==-1000:
2920.               B2.rect.x=1000
2921.
2922.           pygame.display.update()
2923.           fpsClock.tick(FPS)
2924.
2925.       def homeMenu(screen):
2926.           running=True
2927.           BG1=Image("background", 0, 0)
2928.           BG2=Image("background", 1000, 0)
2929.           title=Image("title", 100, 0)
2930.
2931.           play=Button("play", 300, 150)
2932.           Quit=Button("quit", 300, 250)
2933.
2934.           while running:
2935.               spriteList=[BG1, BG2, title, play, Quit]
2936.               drawList(spriteList, screen)
2937.
2938.               for event in pygame.event.get():
2939.                   if event.type == QUIT:
2940.                       pygame.quit()
2941.                       sys.exit()
2942.
2943.                   if event.type == pygame.MOUSEBUTTONDOWN:
2944.                       pos = event.pos
2945.                       if event.button == 1:
2946.                           if checkSelected(play):
2947.                               worldMenu(screen)
2948.                           if checkSelected(Quit):
2949.                               pygame.quit()
2950.                               sys.exit()
2951.
2952.
2953.               update(BG1,BG2)
2954.
2955.
2956.
2957.
2958.       def worldMenu(screen):
2959.
2960.           class World(Entry):
```

```
2961.                 def __init__(self, name, difficulty, time):
2962.                     super().__init__()
2963.
2964.                     self.name=name
2965.                     self.difficulty=difficulty
2966.                     self.time=time
2967.
2968.                     self.path=f"{filepathW}{self.name}.json"
2969.                     self.text=f"Name: {str(self.name)} |
   Difficulty: {str(self.difficulty)} | Time played:
   {str(datetime.timedelta(seconds=round(self.time)))}"
2970.
2971.             def worldSelect(screen):
2972.
2973.                 def getWorldPaths():
2974.                     worldPathList=[]
2975.                     for file in os.listdir(filepathW):
2976.                         if file.endswith(".json"):
2977.                             worldpath=os.path.join(filepathW,
   file)
2978.                             worldPathList.append(worldpath)
2979.
2980.                     return(worldPathList)
2981.
2982.                 def getWorldList():
2983.                     worldPaths=getWorldPaths()
2984.                     worldList=[]
2985.                     for worldPath in worldPaths:
2986.                         with
   open(os.path.expanduser(worldPath)) as file:
2987.                             save = json.load(file)
2988.
2989.                             worldName=save["Name"]
2990.                             time=save["TimePlayed"]
2991.                             difficulty=save["Difficulty"]
2992.
2993.                             world=World(worldName,
   difficulty, time)
2994.                             worldList.append(world)
2995.
2996.                     for index, world in enumerate(worldList):
2997.                         yOffset=(75*index)
2998.                         world.rect.y+=yOffset
2999.
3000.                     return worldList
3001.
3002.                 running=True
3003.                 BG1=Image("background", 0, 0)
```

```
3004.                BG2=Image("background", 1000, 0)
3005.                title=Image("title", 100, 0)
3006.                back=Button("back", 700, 400)
3007.                play=Button("play", 725, 125)
3008.                delete=Button("delete", 700, 225)
3009.
3010.                newWorld=Button("newworld", 675, 300)
3011.                selectWorld=Image("selectworld", 200, 100)
3012.                transBox=Box((600,375), (58,73,135), 128, 50,
   100)
3013.                subtitleBox=Box((600,75), (58,73,135), 256,
   50, 100)
3014.                worldsFull=Text("Unable to create world, max
   four worlds allowed", 20, (255,0,0), 0, 675, 375)
3015.
3016.                worldList=getWorldList()
3017.                worldList
3018.                while running:
3019.
3020.                    spriteList=[BG1, BG2, title, newWorld,
  back, transBox, subtitleBox, selectWorld]+worldList
3021.                    drawList(spriteList, screen)
3022.
3023.                    for event in pygame.event.get():
3024.                        if event.type == QUIT:
3025.                            pygame.quit()
3026.                            sys.exit()
3027.
3028.                        if event.type ==
   pygame.MOUSEBUTTONDOWN:
3029.                            pos = event.pos
3030.                            if event.button == 1:
3031.
3032.                                for world in worldList:
3033.                                    if checkSelected(world):
3034.
3035.                                        for item in
   worldList:
3036.                                            item.chosen=False
3037.
3038.                                        world.chosen=True
3039.
3040.                                    if world.chosen:
3041.                                        if
   checkSelected(play):
3042.                                            game(screen,
   world.name, world.difficulty, world.time)
```

```
                                        if checkSelected(delete):
                                            os.remove(world.path)
                                            worldMenu(screen)

                                    if checkSelected(newWorld):
                                        if len(worldList) < 4:
                                            createWorld(screen)
                                        else:
                                            worldsFull.timer=100

                                    if checkSelected(back):
                                        menu(screen)

                    for world in worldList:
                        if world.chosen:
                            drawList([play, delete], screen)


                    if worldsFull.timer > 0:
                        worldsFull.timer -= 1
                        displayText(worldsFull)

                    update(BG1,BG2)



            def createWorld(screen):

                class NameEntry(Entry):
                    def __init__(self):
                        super().__init__()

                        self.name=""
                        self.difficulty=""
                        self.text=f"Name: {str(self.name)}, Difficulty: {str(self.difficulty)}"

                class Diff(Entry):
                    def __init__(self, text, y):
                        super().__init__()
                        self.text=text
                        self.rect.y = y

                running=True
                BG1=Image("background", 0, 0)
                BG2=Image("background", 1000, 0)
```

```
3088.                 title=Image("title", 100, 0)
3089.                 back=Button("back", 700, 350)
3090.
3091.                 create=Button("create", 700, 250)
3092.                 createWorld=Image("createworld", 200, 100)
3093.                 transBox=Box((600,375), (58,73,135), 128, 50,
   100)
3094.                 subtitleBox=Box((600,75), (58,73,135), 256,
   50, 100)
3095.
3096.                 nameEntry=NameEntry()
3097.                 diffArray=[Diff("Normal", 250),
   Diff("Casual", 325), Diff("Creative", 400),]
3098.
3099.                 while running:
3100.
3101.                     spriteList=[BG1, BG2, title, create,
   back, transBox, subtitleBox, createWorld, nameEntry]+diffArray
3102.                     drawList(spriteList, screen)
3103.
3104.                     for event in pygame.event.get():
3105.                         if event.type == QUIT:
3106.                             pygame.quit()
3107.                             sys.exit()
3108.
3109.                         if event.type ==
   pygame.MOUSEBUTTONDOWN:
3110.
3111.                             if event.button == 1:
3112.
3113.                                 for diff in diffArray:
3114.                                     if checkSelected(diff):
3115.
3116.                                         for item in
   diffArray:
3117.                                             item.chosen=False
3118.
3119.                                         diff.chosen=True
3120.
   nameEntry.difficulty=diff.text
3121.
3122.                                 if checkSelected(nameEntry):
3123.                                     nameEntry.chosen=True
3124.                                 else:
3125.                                     nameEntry.chosen=False
3126.
3127.                                 if checkSelected(create):
```

```
                                        game(screen,
    nameEntry.name, nameEntry.difficulty)

                                    if checkSelected(back):
                                        worldSelect(screen)

                            if event.type == KEYDOWN:
                                if nameEntry.chosen:
                                    if event.key == K_BACKSPACE:
                                        if len(nameEntry.name)>0:
                                            nameEntry.name =
    nameEntry.name[:-1]

                                    else:
                                        nameEntry.name +=
    event.unicode

                        update(BG1,BG2)


                worldSelect(screen)
            homeMenu(screen)
        menu(screen)
```

## References:

I used this website to generate the font: https://fontspool.com/generator/terraria-font

I used photopea to colour the font to yellow: https://www.photopea.com/

Used this source for the background https://wallpapersafari.com/w/dNMT4z

I didn't know how to find all files in a directory with a certain extension to get all world files

https://stackoverflow.com/questions/3964681/find-all-files-in-a-directory-with-extension-txt-in-python

using this as inspiration

```
import os
for file in os.listdir("/mydir"):
    if file.endswith(".txt"):
        print(os.path.join("/mydir", file))
```

I did not know how to create an entry box in pygame for the world creation so I used this website.

https://www.geeksforgeeks.org/how-to-create-a-text-input-box-with-pygame/

To work out how to get the Unicode character of key pressed for entry boxed I used pygame documentation

I obtained player sprite sheet from https://www.deviantart.com/omega7321/art/Terraria-Default-Player-sprite-sheet-637899627

I didn't know how to load spritesheets so I used this video on loading spritesheets:

https://www.youtube.com/watch?v=M6e3_8LHc7A
https://www.youtube.com/watch?v=MYaxPa_eZS0

https://is4-ssl.mzstatic.com/image/thumb/Purple117/v4/ac/85/67/ac856705-3010-f143-05b8-703a027ea907/source/256x256bb.jpg

Took the ore images from here

https://gamevoyagers.com/how-to-get-every-ore-terraria/

block spritesheet: 

procedural gen

https://www.youtube.com/watch?v=Pgt82G4Jxac

https://www.craftycreations.net/wp-content/uploads/2020/08/Bedrock-Block.png

Using this image for the border of the world

https://www.deviantart.com/redballbomb/art/terraria-sprite-sheet-801369142

Tree image:

https://www.pngkit.com/bigpic/u2q8i1o0t4r5r5a9/

I did not know how masks worked in pygame so I watched this video:

https://www.youtube.com/watch?v=uW3Fhe-Vkx4

I am going to use a json file to store all the stats for each tool, however I don't know how to use a json file so I am referring to youtube.

https://www.youtube.com/watch?v=9N6a-VLBa2I

I used this website to check for valid json file.

https://jsonlint.com/

Slime animation

https://c.tenor.com/eMKU7SC46QoAAAAC/slime-terraria.gif

https://forums.terraria.org/index.php?threads/basic-zombie-walking-animation.100745/zombie spritesheet

https://terraria.fandom.com/wiki/Skeleton

took skeleton image from the terraria wiki and then created the animation by overlaying it on top of the player spritesheet and changing it to the skeleton by hand.

For the rest of the images, I used the terraria wiki

 made this my self

https://stackoverflow.com/questions/28015400/how-to-fade-color for day and night cycle

https://stackoverflow.com/questions/4183208/how-do-i-rotate-an-image-around-its-center-using-pygame