BINUS UNIVERSITY
BINUS INTERNATIONAL

Final Project Cover Letter

(Individual Work)

Student Information :

Name: Emanuella Ivana Karunia **Student ID:** 2702323585

Course Code: COMP6047001 **Course Name:** Algorithm and Programing
Class: L1AC **Lecturer:** Jude Joseph Lamug Martinez, MCS

Type Of Assignment : Final Project Report

Submission Date : January 6th, 2024

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Emanuella Ivana Karunia

I. Introduction

For the final project of my Algorithm and Programming class I was tasked to create a project based on the python coding materials I've learned throughout the semester. In this project I've decided to make a simple farming game, making a game in python requires a lot of patience and prior knowledge regarding python as a lot of function and mechanics is needed for a game to work properly. The game I created in this project is an idle farming game where the player can farm crops, chop down wood, along with buying and selling items in game. The game also includes a daytime to nighttime transition and a weather change from sunny to raining.

I chose to create this project mainly because of the aesthetics of the assets and how it will look in the finished product, the artstyle of the pixelated asset seemed adorable and would fit nicely for an idle game I aim to create. The asset I used for this Project is Sprout Lands by Cup Nooble, I used the basic asset pack for this project. The asset pack is quite generous with many assets such as the main player animations, the map and its in-game objects, the house, seeds, and many others. Another reason for me to create this game is because of its simple and relaxing gameplay, I wish to create a simple-idle game where the player could just farm and relax on an island. There are plenty of trials and errors in the process of creating this game because of the many aspects needed in order to shape the game.

As mentioned before, I needed to create many classes and functions to make sure the game can run as smoothly as possible. From animations to the camera group, the tool use and in-game logic, the weather and night transitions all need to be considered when creating this game. Naturally, I've encountered many errors whether it be from misspelling things, or even something as simple as wrong indentation. But this project has helped me learn more things especially in pygame, from simple game animations to in-game logic and function, how to manipulate an image into a game map using tiled, I'm able to learn and understand in the process of creating this project. This project has helped significantly deepen my understanding of python so that I'm able to recognize and debug the small mistakes I made in my code.

II. Algorithm

In order to create a fully functional game we first need a display surface, which will include the background and player later on, for the fondation to display the game. Only after completing this step can we move on to other aspects of the game such as player animation, movements, etc. I'm able to create this window by creating a main.py file that uses screen_width, screen_height information from the settings.py. With this code, I'm able to create a blank display surface for my game and give a title to the display window

main.py > Game > run

```python
import pygame, sys
from settings import *
from level import Level
from support import *

#NOTE TO SELF W TO CHANGE SEED, Q CHANGE TOOL, SPACE USE TOOL, SHIFT TO USE SEED
class Game:
    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode((screen_width,screen_height))
        pygame.display.set_caption('Farm-Fables')
```

(2.1)main.py display surface

settings.py > ...

```python
from pygame.math import Vector2

#screen settings
screen_width = 1280
screen_height = 720
```

(2.2)settings.py screen settings

The main.py file is responsible for running the game and checks whether the player has exited the game or not. I also added a delta time(dt) in main.py to determine the time between each frame so that the in-game movements are able to run smoothly. In another file called level.py, it imports classes and functions of every aspect of the game and compiles all of them in the run() function. It includes the many in-game logic used in this game by calling the functions from every game class accordingly.

```
    def run(self):
        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

            dt = self.clock.tick() / 1000.0
            self.level.run(dt)
            pygame.display.update()

if __name__ == '__main__':
    game = Game()
    game.run()
```

(2.3)main.py exit, dt, & run

Next is the player.py file which is used to determine what the player can do in terms of player animation, movement, and player interaction with the game environment.

player.py > Player

```
import pygame
from settings import *
from support import *
from timer import Timer

class Player(pygame.sprite.Sprite):
    def __init__(self, pos, group, collision_sprites, tree_sprites, interaction, soil_layer,
                 toggle_shop, open_inventory):
        super().__init__(group)
        #import assets
        self.import_assets()
        self.status = 'down_idle'
        self.frame_index = 0
        # General setup
        self.image = self.animations[self.status][self.frame_index]
        self.rect = self.image.get_rect(center = pos)
        self.z = layers['main']
        # Movement attributes
        self.direction = pygame.math.Vector2()
        # Default position on the center
        self.pos = pygame.math.Vector2(self.rect.center)
        self.speed = 200
```

(2.4)player.py init function part 1

The player init function is set to call the player animation, movement, player interaction, and to store dictionaries & lists used for some of the player interaction and movement.

```
        #collison
        self.collision_sprites = collision_sprites
        self.hitbox = self.rect.copy().inflate((-126,-70))
        #timer
        self.timers = {
            'tool use': Timer(350,self.use_tool),
            'tool switch': Timer(200),
            'seed use': Timer(350,self.use_seed),
            'seed switch': Timer(200)
        }
        #tools
        self.tools = ['hoe', 'axe', 'water']
        self.tool_index = 0
        self.selected_tool = self.tools[self.tool_index]
        #seeds
        self.seeds = ['corn', 'tomato']
        self.seed_index = 0
        self.selected_seed = self.seeds[self.seed_index]
        #inventory
        self.item_inventory = {
            'wood': 0,
            'apple': 0,
            'corn': 0,
            'tomato': 0
        }
        #seed inventory
        self.seed_inventory = {
            'corn': 10,
            'tomato': 10
        }
```

(2.5)player.py init function part 2

```
        self.money = 100
        #interaction
        self.tree_sprites = tree_sprites
        self.interaction = interaction
        self.sleep = False
        self.soil_layer = soil_layer
        self.toggle_shop = toggle_shop
        #inventory
        self.open_inventory = open_inventory
        #sfx
        self.watering = pygame.mixer.Sound('../audio/water.mp3')
        self.watering.set_volume(0.1)

```

(2.6)player.py init function part 3

First is the player animations, this can be done by accessing the asset folder for the character. Inside the character folder there's more folders labeled with the character status such as down, up, left, right, which contains a sequence of images labeled from 0 to 3. The program will then loop the images contained in the folder to create a smooth animation depending on the player status.

```
    #make sure to import the right asset
    def import_assets(self):
        self.animations = {'up': [],'down': [],'left': [],'right': [],
                           'right_idle':[],'left_idle':[],'up_idle':[],'down_idle':[],
                           'right_hoe':[],'left_hoe':[],'up_hoe':[],'down_hoe':[],
                           'right_axe':[],'left_axe':[],'up_axe':[],'down_axe':[],
                           'right_water':[],'left_water':[],'up_water':[],'down_water':[]}

        for animation in self.animations.keys():
            full_path = '../graphics/character/' + animation
            self.animations[animation] = import_folder(full_path)

    #make the image animate
    def animate(self,dt):
        self.frame_index += 4 * dt
        if self.frame_index >= len(self.animations[self.status]):
            self.frame_index = 0

        self.image = self.animations[self.status][int(self.frame_index)]
```

(2.7)player.py import_assets and animate function

The player status is used to tell the program what the player is currently doing so the program could find the corresponding folder which contains the correct status for the program to be able to animate. This status could be obtained by checking the player input in the input function or by using the get_status function which checks if the player is not moving/idle or if the player is using a tool or not.

```
106     def input(self):
107         keys = pygame.key.get_pressed()
108         if not self.timers['tool use'].active and not self.sleep:
109             #direction input
110             # Vertical axis
111             if keys[pygame.K_UP]:
112                 self.direction.y = -1
113                 self.status = 'up'
114             elif keys[pygame.K_DOWN]:
115                 self.direction.y = 1
116                 self.status = 'down'
117             else:
118                 self.direction.y = 0
119             # Horizontal axis
120             if keys[pygame.K_RIGHT]:
121                 self.direction.x = 1
122                 self.status = 'right'
123             elif keys[pygame.K_LEFT]:
124                 self.direction.x = -1
125                 self.status = 'left'
126             else:
127                 self.direction.x = 0
128             #tool use
129             if keys[pygame.K_SPACE]:
130                 #timer for tool use
131                 self.timers['tool use'].activate()
132                 self.direction = pygame.math.Vector2()
133                 #making sure the frame starts at 0
134                 self.frame_index = 0
```

(2.8)player.py input function part 1

```
135             #changing tools
136             #make the output a single number(the img file is a number from 0-3)
137             if keys[pygame.K_q] and not self.timers['tool switch'].active:
138                 self.timers['tool switch'].activate()
139                 #go through the tool dict to access each tool
140                 self.tool_index += 1
141                 #loop through the len of tools
142                 if self.tool_index < len(self.tools):
143                     self.tool_index = self.tool_index
144                 else :
145                     self.tool_index = 0
146                 self.selected_tool = self.tools[self.tool_index]
147 
148             #seed use
149             if keys[pygame.K_LSHIFT]:
150                 #timer for seed use
151                 self.timers['seed use'].activate()
152                 self.direction = pygame.math.Vector2()
153                 #making sure the frame starts at 0
154                 self.frame_index = 0
155             #changing seeds
156             #make the output a single number(the img file is a number from 0-3)
157             if keys[pygame.K_w] and not self.timers['seed switch'].active:
158                 self.timers['seed switch'].activate()
159                 #go through the seed dict to access each tool
160                 self.seed_index += 1
161                 #loop through the len of tools
162                 if self.seed_index < len(self.seeds):
163                     self.seed_index = self.seed_index
164                 else :
```

(2.9)player.py input function part 2

The input function is used to check what keys the player is pressing to control the movement and interaction of the player with the environment. In this case if the player pressed up,down,left or right, it would move the character to the corresponding direction and update the status of the player to display the correct animation. The space bar is used to use the tool, the left shift key is used to use the seed, the q key is used to change the tool, the w key is used to change the seed, the i key is used to view the inventory, and the enter key is used for the player interaction with the bed and trader if they stand at a certain area in the map(beside the bed or in front of the trader). In the get_status function, it's used if the player isn't moving, it will change the player status to idle by accessing the direction the player was previously facing and adding a '_idle', it will do the same thing with the 'tool use' but by adding the selected_tool instead, for example if the player is facing the left and is using the hoe tool, the status will be updated to 'left_hoe' in the update function to access the right animation for the player interaction.

```
                else :
                    self.seed_index = 0
                self.selected_seed = self.seeds[self.seed_index]
            #restart day & trader
            if keys[pygame.K_RETURN]:
                collided_interaction_sprite = pygame.sprite.spritecollide(self, self.interaction,False)
                if collided_interaction_sprite:
                    if collided_interaction_sprite[0].name == 'Trader':
                        self.toggle_shop()
                    else:
                        self.status = 'left_idle'
                        self.sleep = True
            #open inventory
            if keys[pygame.K_i]:
                self.open_inventory()


    def get_status(self):
        #check if player is idle
        if self.direction.magnitude() == 0:
            #make sure there's only 1 idle in the status
            self.status = self.status.split('_')[0] + '_idle'
            #using tools
        if self.timers['tool use'].active:
            self.status = self.status.split('_')[0] + '_' + self.selected_tool
    #make sure to not loop the tool use forever
```

(2.10)player.py input part 3 & get_status function

After establishing the player movement and animation, we need to add the map and collisions, the player needs to be able to collide with the objects on the map such as the trader, house walls, this collision function is also needed so that the player can't walk straight out of the map. This has already been done in the tmx data for the map which could be called in the level.py in the setup function.

(2.11)collision layer in the map.tmx data

level.py > ...

```python
import pygame
import pytmx
from pytmx.util_pygame import load_pygame
from settings import *
from player import Player
from overlay import Overlay
from sprites import Generic, Water, WildFlower, Tree, Interaction, Particle
from support import *
from transition import Transition
from soil import SoilLayer
from sky import Rain, Sky
from random import randint
from menu import Menu
from inventory import Inventory

class Level:
    def __init__(self):
      #get display surface
      self.display_surface = pygame.display.get_surface()
      #sprite groups
      self.all_sprites = CameraGroup()
      self.collision_sprites = pygame.sprite.Group()
      self.tree_sprites = pygame.sprite.Group()
      self.interaction_sprite = pygame.sprite.Group()

      self.soil_layer = SoilLayer(self.all_sprites, self.collision_sprites)
      self.setup()
      self.overlay = Overlay(self.player)
      self.transition = Transition(self.reset,self.player)
```

(2.12)level.py init function part 1

```python
    #sky
    self.rain = Rain(self.all_sprites)
    self.raining = randint(0,10) > 7
    self.soil_layer.raining = self.raining
    self.sky = Sky()

    #shop
    self.menu = Menu(self.player, self.toggle_shop)
    self.shop_active = False
    #inventory
    self.inventory = Inventory(self.player, self.open_inventory)
    self.inventory_active = False
    #sfx
    self.success = pygame.mixer.Sound('../audio/success.wav')
    self.success.set_volume(0.3)
    self.bg_music = pygame.mixer.Sound('../audio/music.mp3')
    self.bg_music.play(loops = -1)

def setup(self):
    tmx_data = load_pygame('../data/map.tmx')
    #house
    for layer in ['HouseFloor', 'HouseFurnitureBottom']:
      for x, y, surf in tmx_data.get_layer_by_name(layer).tiles():
        Generic((x * tile_size, y * tile_size),surf,self.all_sprites,layers['house bottom'])

    for layer in ['HouseWalls', 'HouseFurnitureTop']:
      for x, y, surf in tmx_data.get_layer_by_name(layer).tiles():
        Generic((x * tile_size, y * tile_size),surf,self.all_sprites,layers['main'])
```

(2.13)level.py init function part 2 and setup function part 1

```
        #fence
            for x, y, surf in tmx_data.get_layer_by_name('Fence').tiles():
                Generic((x * tile_size, y * tile_size),surf,[self.all_sprites, self.collision_sprites],layers['main'])
        #water
            water_frames = import_folder('../graphics/water')
            for x, y, surf in tmx_data.get_layer_by_name('Water').tiles():
                Water((x * tile_size, y * tile_size),water_frames, self.all_sprites)
        #trees
            for obj in tmx_data.get_layer_by_name('Trees'):
                Tree(pos =(obj.x, obj.y),
                     surf = obj.image,
                     groups = [self.all_sprites, self.collision_sprites,self.tree_sprites],
                     name = obj.name,
                     player_add = self.player_add)
        #wildflowers
            for obj in tmx_data.get_layer_by_name('Decoration'):
                WildFlower((obj.x, obj.y), obj.image, [self.all_sprites, self.collision_sprites])
        #collision tiles
            for x, y, surf in tmx_data.get_layer_by_name('Collision').tiles():
                Generic((x * tile_size, y * tile_size),pygame.Surface((tile_size,tile_size)),self.collision_sprites)
        #Player
            for obj in tmx_data.get_layer_by_name('Player'):
                if obj.name == 'Start':
                    self.player = Player(pos = (obj.x,obj.y),
                                         group = self.all_sprites,
                                         collision_sprites = self.collision_sprites,
                                         tree_sprites = self.tree_sprites,
                                         interaction = self.interaction_sprite,
                                         soil_layer = self.soil_layer,
                                         toggle_shop = self.toggle_shop,
```

(2.14)level.py setup function part 2

```
                                         toggle_shop = self.toggle_shop,
                                         open_inventory = self.open_inventory)
                if obj.name == 'Bed':
                    Interaction((obj.x,obj.y),(obj.width, obj.height),self.interaction_sprite, obj.name)
                if obj.name == 'Trader':
                    Interaction((obj.x,obj.y),(obj.width, obj.height),self.interaction_sprite, obj.name)

            Generic(pos = (0,0),
                    surf = pygame.image.load('../graphics/world/ground.png').convert_alpha(),
                    groups= self.all_sprites,
                    z = layers['ground'])
```

(2.15)level.py setup function part 3

The setup function in level.py displays the objects in the map according to the layers dictionary set in settings.py and the data stored in the map.tmx data. The layers dictionary states which objects should be drawn in a certain layer to ensure that every object is visible to the player. If a layer overlaps, the objects drawn in the previous layer might not be completely visible because the object in the layer on top of it is in the way, the layers dictionary ensures that no layer is overlapping so that everything is clearly visible to the player. The tmx data is used to create the layers in the map, inside map.tmx there are many layers with clear names to indicate which layer each object belongs to. It's also used to block off certain areas so that the player couldn't pass through the walls of the house, walk over the trader, or walk out of the map, and it's used to tell

which area is farmable and therefore could be turned into a soil patch, watered,and planted with a seed.

After creating the map and player, it's important that we make a camera group or else we can't always see where the player is located and what they're doing. What a camera group does is essentially create a rectangle the size of the game window that follows the player around, which could shift the map according to where the player is moving. For example the code below includes the player position which will be used to constantly shift the camera group so that we can always see where the player is moving and what they're doing.

```
162  class CameraGroup(pygame.sprite.Group):
163      def __init__(self):
164          super().__init__()
165          self.display_surface = pygame.display.get_surface()
166          #make the camera follow the player
167          self.offset = pygame.math.Vector2()
168
169      def custom_draw(self,player):
170          #how much the sprites is shifting + ensure that the player is always at the center
171          self.offset.x = player.rect.centerx - screen_width/2
172          self.offset.y = player.rect.centery - screen_height/2
173          #make sure an image is in the right position(layer)
174          for layer in layers.values():
175              #make 3d effect when the player is behind or in front of an object
176              for sprite in sorted(self.sprites(), key = lambda sprite : sprite.rect.centery):
177                  if sprite.z == layer:
178                      offset_rect = sprite.rect.copy()
179                      offset_rect.center -= self.offset
180                      self.display_surface.blit(sprite.image, offset_rect)
```

(2.11)level.py class CameraGroup

(2.16)Farmable area in the map.tmx data

Moving on to the player interaction, in this game the player could farm, chop wood & collect apples, sleep, and sell or buy from the trader. In the farming method we first need to check if the area is farmable or not, we need to make sure it doesn't collide with the objects in the map and that it needs to be on the area where there's nothing on it. As shown on the picture above, there's already an area in the map.tmx blocked off as the farmable area in the map.

soil.py > SoilLayer > plant_seed

```python
import pygame
from settings import *
from support import *
from pytmx.util_pygame import load_pygame
from random import choice

class SoilTile(pygame.sprite.Sprite):
    def __init__(self, pos, surf, groups):
        super().__init__(groups)
        self.image = surf
        self.rect = self.image.get_rect(topleft = pos)
        self.z = layers['soil']
```

(2.17)soil,py SoilTile class

This class is used to set where the soil image should be drawn, in the 'Soil' layer and creates a rectangle for the image itself. Next we need to check if the area we want to farm in is a part of the farmable area or not, this is done in the SoilLayer class.

```python
class SoilLayer:
    def __init__(self, all_sprites, collision_sprites):
        #sprite groups
        self.all_sprites = all_sprites
        self.collision_sprites = collision_sprites
        self.soil_sprites = pygame.sprite.Group()
        self.water_sprites = pygame.sprite.Group()
        self.plant_sprites = pygame.sprite.Group()
        #graphics
        self.soil_surfs = import_folder_dict('../graphics/soil/')
        self.water_surfs = import_folder('../graphics/soil_water/')

        self.create_soil_grid()
        self.create_hit_rects()
        #sfx
        self.hoe_sound = pygame.mixer.Sound('../audio/hoe.wav')
        self.hoe_sound.set_volume(0.1)
        self.plant_sound = pygame.mixer.Sound('../audio/plant.wav')
        self.plant_sound.set_volume(0.2)

    def create_soil_grid(self):
        ground = pygame.image.load('../graphics/world/ground.png')
        h_tiles, v_tiles = ground.get_width()// tile_size, ground.get_height()// tile_size
        self.grid = [[[] for col in range(h_tiles)] for row in range(v_tiles)]
        for x, y, surf in load_pygame('../data/map.tmx').get_layer_by_name('Farmable').tiles():
            self.grid[y][x].append('F')
```

(2.18)soil.py SoilLayer init and create_soil_grid function

In the create_soil_grid function, it defines the ground image and loads the data from map.tmx specifically the 'Farmable' layer by using the get_layer_by_name function in pygame. It gets the coordinates of the player by dividing the ground width and height with the tile size(64 px). Then if the area is farmable, it appends the string 'F'. Then if the farmable ground is being hit, it appends a 'X' string to the grid in the get_hit function. The create_hit_rects creates a rectangle for the soil patch once a tile in the farmable area gets hit with the hoe. Once 'X' is appended in the tile, the tile could then be turned into a soil patch by importing one of the images in the soil folder. The get_hit function also waters all the soil patches if the weather is raining. In the create_soil_tiles function, it checks the surrounding tiles that just got hit and then updates the soil patch accordingly so that if we create 4 soil patches next to each other, it'll just be one big soil patch rather than four individual soil patches created next to each other. This function needs a lot of if statements as there are many possibilities for the soil patch image.

```
82      def create_hit_rects(self):
83          self.hit_rects = []
84          for index_row, row in enumerate(self.grid):
85              for index_col, cell in enumerate(row):
86                  if 'F' in cell:
87                      x = index_col * tile_size
88                      y = index_row * tile_size
89                      rect = pygame.Rect(x,y, tile_size,tile_size)
90                      self.hit_rects.append(rect)
91
92      def get_hit(self, point):
93          for rect in self.hit_rects:
94              if rect.collidepoint(point):
95                  self.hoe_sound.play()
96                  x = rect.x // tile_size
97                  y = rect.y // tile_size
98                  if 'F' in self.grid[y][x]:
99                      self.grid[y][x].append('X')
100                     self.create_soil_tiles()
101                     if self.raining:
102                         self.water_all()
```

(2.19)soil.py create_hit_rects & get_hit function

```
158     def create_soil_tiles(self):
159         self.soil_sprites.empty()
160         for index_row, row in enumerate(self.grid):
161             for index_col, cell in enumerate(row):
162                 if 'X' in cell:
163                     #check the tiles surrounding the soil tile
164                     t = 'X' in self.grid[index_row - 1][index_col]
165                     b = 'X' in self.grid[index_row + 1][index_col]
166                     r = 'X' in row[index_col + 1]
167                     l = 'X' in row[index_col - 1]
168                     tile_type = 'o'
169 
170                     #change soil img if all sides are soil
171                     if all((t,b,r,l)): tile_type = 'x'
172                     #change img if any of the horizontal sides are soil
173                     if l and not any((t,r,b)): tile_type = 'r'
174                     if r and not any((t,l,b)): tile_type = 'l'
175                     if l and r and not any((t,b)): tile_type = 'lr'
176                     #change img if any of the vertical sides are soil
177                     if t and not any((l,r,b)): tile_type = 'b'
178                     if b and not any((t,l,r)): tile_type = 't'
179                     if t and b and not any((r,l)): tile_type = 'tb'
180                     #change img for the corners of the soil
181                     if b and r and not any((t,l)): tile_type = 'tl'
182                     if b and l and not any((t,r)): tile_type = 'tr'
183                     if t and r and not any((b,l)): tile_type = 'bl'
184                     if t and l and not any((b,r)): tile_type = 'br'
```

(2.20)soil.py create_soil_tiles part 1

```
182                     # T shapes
183                     if all((t,b,r)) and not l: tile_type = 'tbr'
184                     if all((t,b,l)) and not r: tile_type = 'tbl'
185                     if all((l,r,t)) and not b: tile_type = 'lrt'
186                     if all((l,r,b)) and not t: tile_type = 'lrb'
187                     #middle parts :-)(EAAUUURGHHHHHH)
188                     if all((b,l,r)) and not t: tile_type = 'tm'
189                     if all((t,l,r)) and not b: tile_type = 'bm'
190                     if all((t,b,r)) and not l: tile_type = 'lm'
191                     if all((t,b,l)) and not r: tile_type = 'rm'
192 
193                     SoilTile(pos = (index_col * tile_size, index_row * tile_size),
194                              surf = self.soil_surfs[tile_type],
195                              groups = [self.all_sprites,self.soil_sprites])
```

(2.21)soil.py create_soil_tiles part 2

Next is the water function, the player could water a soil patch by changing the tool to a watering can and use it by pressing the spacebar on a soil patch. If a soil patch is watered, it will append the string ‘W’ to the soil patch then it will update the soil patch surface by adding another image on top of it from the soil_water folder which is accessed by the water_surfs variable. The remove_water function is used at the reset in level.py after the player goes to sleep.

```
104     def water(self,target_pos):
105         for soil_sprite in self.soil_sprites.sprites():
106             if soil_sprite.rect.collidepoint(target_pos):
107                 x = soil_sprite.rect.x //tile_size
108                 y = soil_sprite.rect.y //tile_size
109                 self.grid[y][x].append('W')
110                 #water sprite
111                 WaterTile(pos = soil_sprite.rect.topleft,
112                           surf = choice(self.water_surfs),
113                           groups = [self.all_sprites,self.water_sprites])
114     def water_all(self):
115         for index_row, row in enumerate(self.grid):
116             for index_col, cell in enumerate(row):
117                 if 'X' in cell and 'W' not in cell:
118                     cell.append('W')
119                     x = index_col * tile_size
120                     y = index_row * tile_size
121                     WaterTile(pos = (x,y),
122                               surf =choice(self.water_surfs),
123                               groups = [self.all_sprites, self.water_sprites])
124     def remove_water(self):
125         #destroy water sprites
126         for sprite in self.water_sprites.sprites():
127             sprite.kill()
128         #clean grid
129         for row in self.grid:
130             for cell in row:
131                 if 'W' in cell:
132                     cell.remove('W')
```

(2.22)soil.py water, water_all, remove_water function

After the soil patch is watered, we can plant the seed. The seed could be planted after we made the soil layer but it will only grow after the soil patch is watered. To check if a soil patch is watered, a check_watered function is used. It checks if there's a 'W' in the tile then it will return is_watered.

```
134     def check_watered(self, pos):
135         x = pos[0] // tile_size
136         y = pos[1] // tile_size
137         cell = self.grid[y][x]
138         is_watered = 'W' in cell
139         return is_watered
140 
141     def plant_seed(self, target_pos, seed):
142         for soil_sprite in self.soil_sprites.sprites():
143             if soil_sprite.rect.collidepoint(target_pos):
144                 self.plant_sound.play()
145                 x = soil_sprite.rect.x // tile_size
146                 y = soil_sprite.rect.y // tile_size
147                 if 'P' not in self.grid[y][x]:
148                     self.grid[y][x].append('P')
149                     Plant(seed, [self.all_sprites, self.plant_sprites, self.collision_sprites], soil_sprite,
150                           self.check_watered)
151 
152     def update_plants(self):
153         for plant in self.plant_sprites.sprites():
154             plant.grow()
```

(2.23)soil.py check_watered, plant_seed, update_plant function

If the player planted a seed, a sound will play and a sprout will appear on top of the watered soil patch. After the player sleeps, the plant will grow.

```
class Plant(pygame.sprite.Sprite):
    def __init__(self, plant_type, groups, soil, check_watered):
        super().__init__(groups)
        self.plant_type = plant_type
        self.frames = import_folder(f'../graphics/fruit/{plant_type}')
        self.soil = soil
        self.check_watered = check_watered
        self.age = 0
        self.max_age = len(self.frames) - 1
        self.grow_speed = Grow_speed[plant_type]
        self.harvestable = False
        #sprite
        self.image = self.frames[self.age]
        if plant_type == 'corn': self.y_offset = -16
        else: self.y_offset = -8
        self.rect = self.image.get_rect(midbottom = soil.rect.midbottom + pygame.math.Vector2(0,self.y_offset))
        self.z = layers['ground plant']
    def grow(self):
        if self.check_watered(self.rect.center):
            self.age += self.grow_speed
            if int(self.age) > 0:
                self.z = layers['main']
                self.hitbox = self.rect.copy().inflate(-26,-self.rect.height *0.4)
            #making sure the plant age don't go over the max age
            if self.age >= self.max_age:
                self.age = self.max_age
                self.harvestable = True
            self.image = self.frames[int(self.age)]
            self.rect = self.image.get_rect(midbottom = self.soil.rect.midbottom + pygame.math.Vector2(0,self.y_offse
```

(2.24)soil.py Plant class

The plant class specifies the plant attribute, it sets the image of the plant, the age, max age, where it should sit in the soil patch and in the grow function it creates a hitbox for the plant. The hitbox is used so that while the player could walk over a plant that's still a sprout, the player can't walk over a growing plant so the hitbox is there to ensure that the player will collide with a growing plant. The grow function also updates the plant image according to the growth speed of each plant and when the plant image could no longer be updated, the plant is ready to be harvested. When the player harvests the plant a white particle will appear and the selected_seed(corn/tomato) will be added to their inventory. The particle will appear after the player collects their harvest, collects apples, or when they chop down a tree.

```
class Particle(Generic):
    def __init__(self,pos,surf,groups,z,duration = 200):
        super().__init__(pos,surf,groups, z)
        self.start_time = pygame.time.get_ticks()
        self.duration = duration
        #white surf
        mask_surf = pygame.mask.from_surface(self.image)
        new_surf = mask_surf.to_surface()
        new_surf.set_colorkey((0,0,0))
        self.image = new_surf

    def update(self,dt):
        current_time = pygame.time.get_ticks()
        if current_time - self.start_time > self.duration:
            self.kill()
```

(2.25)sprites.py particle class

The player could also interact with the tree by chopping it down to collect wood and apples. This tree class is created in the sprites.py file. The tree class inherited the values from the generic class, also from the sprites.py file.

```
1   import pygame
2   from settings import *
3   from random import randint, choice
4   from timer import Timer
5
6   class Generic(pygame.sprite.Sprite):
7       def __init__(self, pos, surf, groups, z = layers['main']):
8           super().__init__(groups)
9           self.image = surf
10          self.rect = self.image.get_rect(topleft = pos)
11          self.z = z
12          self.hitbox = self.rect.copy().inflate(-self.rect.width * 0.2, -self.rect.height * 0.75)
```

(2.26)sprites.py Generic class

The tree is able to produce apples and could be cut down for wood, once the tree is dead the tree image is updated to a tree stump.

```
60  class Tree(Generic):
61      def __init__(self, pos, surf, groups, name, player_add):
62          super().__init__(pos, surf, groups)
63          #tree attributes
64          self.health = 5
65          self.alive = True
66          stump_tree = f'../graphics/stumps/{"small" if name == "Small" else "large"}.png'
67          self.stump_surf = pygame.image.load(stump_tree).convert_alpha()
68          #apples
69          self.apple_surf = pygame.image.load('../graphics/fruit/apple.png')
70          self.apple_pos = Apple_pos[name]
71          self.apple_sprites = pygame.sprite.Group()
72          self.create_fruit()
73          self.player_add = player_add
74          #copy of the original image
75          self.original_surf = surf.copy()
76          #sounds
77          self.axe_sound = pygame.mixer.Sound('../audio/axe.mp3')
78      def damage(self):
79          #dmg to tree
80          self.health -= 1
81          #play axe sound
82          self.axe_sound.play()
83          #remove an apple
84          if len(self.apple_sprites.sprites())>0:
85              random_apple = choice(self.apple_sprites.sprites())
86              Particle(pos= random_apple.rect.topleft,
87                       surf = random_apple.image,
88                       groups = self.groups()[0],
89                       z = layers['fruit'])
```

(2.27)sprites.py Tree class part 1

```
            self.player_add('apple')
            random_apple.kill()
    def check_death(self):     #check if the tree is dead
        if self.health <= 0:
            Particle(self.rect.topleft, self.image, self.groups()[0], layers['fruit'],duration=300)
            self.image = self.stump_surf
            self.rect = self.image.get_rect(midbottom = self.rect.midbottom)#make tree stump is in same place
            self.hitbox = self.rect.copy().inflate(-10,-self.rect.height * 0.6)
            self.alive = False
            self.player_add('wood')
    def update(self,dt):
        if self.alive:
            self.check_death()
    def regrow(self):            #respawn dead tree
        if self.alive == False:
            self.health = 5
            self.alive = True
            self.image = self.original_surf.copy()
            self.rect = self.image.get_rect(midbottom=self.rect.midbottom)
            self.hitbox = self.rect.copy().inflate(-10, -self.rect.height * 0.6)
    def create_fruit(self):
        for pos in self.apple_pos:
            if randint(0,10) < 2:
                x = pos[0] + self.rect.left
                y = pos[1] + self.rect.top
                Generic(pos = (x,y),
                        surf = self.apple_surf,
                        groups = [self.apple_sprites,self.groups()[0]],
                        z = layers['fruit'])
```

(2.28)sprites.py Tree class

The apple is drawn on the surface of the tree, the max amount of apples per tree depends on whether the tree is a small or medium tree. The medium tree has more surface area than the small tree, therefore the amount of apples they could produce are more than the small tree. Each tree has a label to indicate if it's a small or a medium tree, this data is stored in the map.tmx data and the apple position is stored in the settings.py which contains the coordinates of the small and medium trees. In the tree class, the tree has a health bar of 5 which means after the player has hit the tree with an axe 5 times, the tree will become a tree stump and the player would have wood added to their inventory. After the player hits the tree once, one of the apples in the tree will disappear with a white particle in its place for a couple seconds then the apple will be added to the player inventory. The tree will regrow with the regrow function after the player sleeps, the regrow function replaces the tree stump image with a copy of the original tree image and the tree would regrow their apples again.

The player could also sleep and interact with the trader to sell or buy items(seeds). For the sleeping mechanism, the player could stand next to the bed near the window and press enter to sleep. Once they're sleeping, the player status would be updated to left and they wouldn't be able to move until the daytime transition is over. In player.py, if the player is beside the bed then self.sleep becomes true.

```python
def reset(self):
    #plants
    self.soil_layer.update_plants()
    #soil
    self.soil_layer.remove_water()
    #randomize rain
    self.raining = randint(0,10) > 3
    self.soil_layer.raining = self.raining
    if self.raining:
        self.soil_layer.water_all()
    #apple on tree
    for tree in self.tree_sprites.sprites():
        for apple in tree.apple_sprites.sprites():
            apple.kill()
        tree.create_fruit()
        tree.regrow()
    #sky
    self.sky.start_color = [255,255,255]
```

(2.29)level.py reset function

```python
#restart day & trader
if keys[pygame.K_RETURN]:
    collided_interaction_sprite = pygame.sprite.spritecollide(self, self.interaction,False)
    if collided_interaction_sprite:
        if collided_interaction_sprite[0].name == 'Trader':
            self.toggle_shop()
        else:
            self.status = 'left_idle'
            self.sleep = True
```

(2.30)player.py input function

Other than the sleeping mechanism, if the player is standing in front of the trader and pressed enter then they can access the buy or sell menu.

menu.py > Menu > input

```python
import pygame
from settings import *
from timer import Timer

class Menu:
    def __init__(self, player, toggle_menu):
        self.player = player        #general setup
        self.toggle_menu = toggle_menu
        self.display_surface = pygame.display.get_surface()
        self.font = pygame.font.Font('../font/LycheeSoda.ttf',30)
        self.width = 400        #option
        self.space = 10
        self.padding = 8
        self.options = list(self.player.item_inventory.keys()) + list(self.player.seed_inventory.keys()) #entries
        self.sell_border = len(self.player.item_inventory) - 1
        self.setup()
        self.index = 0        #movement
        self.timer = Timer(200)
    def display_money(self):
        text_surf = self.font.render(f'Currency: ${self.player.money}', False, 'Brown')
        text_rect = text_surf.get_rect(midbottom = (screen_width / 2, screen_height - 40))
        pygame.draw.rect(self.display_surface, 'White', text_rect.inflate(10,10),0,6)
        self.display_surface.blit(text_surf,text_rect)
    def setup(self):
        self.text_surfs = []
        self.total_height = 0
        for item in self.options:
            text_surf = self.font.render(item, False, 'Brown')
            self.text_surfs.append(text_surf)
            self.total_height += text_surf.get_height() + (self.padding * 2)
```

```
        self.total_height += (len(self.text_surfs)- 1) * self.space
        self.menu_top = screen_height /2 - self.total_height /2
        self.left = screen_width/2 - self.width/2
        self.main_rect = pygame.Rect(self.left, self.menu_top, self.width, self.total_height)
        self.buy_text = self.font.render('buy', False, 'Green')         # buy/sell text
        self.sell_text = self.font.render('sell', False, 'Red')
    def input(self):
        keys = pygame.key.get_pressed()
        self.timer.update()
        if keys[pygame.K_ESCAPE]:
            self.toggle_menu()
        if not self.timer.active:
            if keys[pygame.K_UP]:
                self.index -= 1
                self.timer.activate()
            if keys[pygame.K_DOWN]:
                self.index += 1
                self.timer.activate()
            if keys[pygame.K_SPACE]:
                self.timer.activate()
                current_item = self.options[self.index]     # get item
                if self.index <= self.sell_border:    # sell or buy the item
                    if self.player.item_inventory[current_item] > 0:
                        self.player.item_inventory[current_item] -= 1
                        self.player.money += sale_prices[current_item]
                else:
                    seed_price = purchase_prices[current_item]
                    if self.player.money >= seed_price:
                        self.player.seed_inventory[current_item] += 1
                        self.player.money -= purchase_prices[current_item]
        if self.index < 0:      # clamp the values
            self.index = len(self.options) - 1
        if self.index > len(self.options) - 1:
            self.index = 0
    def show_entry(self, text_surf, amount, top, selected):
        bg_rect = pygame.Rect(self.main_rect.left, top, self.width, text_surf.get_height() + self.padding * 2)
        pygame.draw.rect(self.display_surface, 'White', bg_rect, 0, 6)
        text_rect = text_surf.get_rect(midleft = (self.main_rect.left + 20, bg_rect.centery))
        self.display_surface.blit(text_surf, text_rect)
        amount_surf = self.font.render(str(amount), False, 'Brown') #amount
        amount_rect = amount_surf.get_rect(midright = (self.main_rect.right - 20,bg_rect.centery))
        self.display_surface.blit(amount_surf, amount_rect)
        if selected: #selected
            pygame.draw.rect(self.display_surface, 'Brown', bg_rect, 4, 4)
            if self.index <= self.sell_border: #sell things
                pos_rect = self.sell_text.get_rect(midleft = (self.main_rect.left + 150, bg_rect.centery))
                self.display_surface.blit(self.sell_text,pos_rect)
            else: #buy things
                pos_rect = self.buy_text.get_rect(midleft = (self.main_rect.left + 150, bg_rect.centery))
                self.display_surface.blit(self.buy_text,pos_rect)
    def update(self):
        self.input()
        self.display_money()
        for text_index, text_surf in enumerate(self.text_surfs):
            top = self.main_rect.top + text_index * (text_surf.get_height() + (self.padding * 2) + self.space)
            amount_list = list(self.player.item_inventory.values()) + list(self.player.seed_inventory.values())
            amount = amount_list[text_index]
            self.show_entry(text_surf, amount, top, self.index == text_index)
```

(2.31)menu.py class Menu

The menu.py file contains an option for the player to sell items which is accessed from the player inventory in player and able to buy items in the seed inventory. This freezes the game window and the player is able to browse through the item index by using the up or down key and they're able to buy or sell items using the spacebar. The selected item will be highlighted by a brown border surrounding a certain item index and the player is able to buy or sell the item based on the buy or sell text beside the item name. If a player sells something, the item will be redacted from their inventory and a certain number of money will be added to their inventory. If a player buys

something from the trader, a certain amount of money is redacted from their inventory and their selected item will be added to their inventory. The prices of each item is determined through a list in settings.py. To exit the menu function, the player needs to press the esc key.

```python
sale_prices = {
    'wood': 4,
    'apple': 2,
    'corn': 10,
    'tomato': 20
}

purchase_prices = {
    'corn': 4,
    'tomato': 5
}
```

(2.32)settings.py prices

The inventory class uses the same logic as the menu.py file, except it doesn't require a specific location to call the function. This function could be used anywhere in the map as long as the player presses the i key, the inventory could only be used to see what the player currently has. The inventory displays the items the player has and how much money they have. This function also can be closed using the esc key.

inventory.py > Inventory > __init__

```python
import pygame
from settings import screen_width, screen_height

class Inventory:
    def __init__(self, player, open_inventory):
        self.player = player        # General setup
        self.display_surface = pygame.display.get_surface()
        self.font = pygame.font.Font('../font/LycheeSoda.ttf', 30)
        self.width = 400        # Option
        self.space = 10
        self.padding = 8
        self.open_inventory = open_inventory        # Toggle
    def display_money(self):
        text_surf = self.font.render(f'Currency: ${self.player.money}', False, 'Brown')
        text_rect = text_surf.get_rect(midbottom=(screen_width / 2, screen_height - 40))
        pygame.draw.rect(self.display_surface, 'White', text_rect.inflate(10, 10), 0, 6)
        self.display_surface.blit(text_surf, text_rect)
    def display_inventory(self):
        title_surf = self.font.render("Inventory", False, 'Brown')
        title_rect = title_surf.get_rect(midtop=(screen_width / 2, 50))
        self.display_surface.blit(title_surf, title_rect)
        top = title_rect.bottom + 20
        for item, amount in self.player.item_inventory.items():
            text_surf = self.font.render(f'{item}: {amount}', False, 'Brown')
            bg_rect = pygame.Rect((screen_width / 2) - 150, top, 300, text_surf.get_height())
            pygame.draw.rect(self.display_surface, 'White', bg_rect, 0, 6)
            text_rect = text_surf.get_rect(center=(screen_width / 2, bg_rect.centery))
            self.display_surface.blit(text_surf, text_rect)
            top += text_surf.get_height() + 10
        for item, amount in self.player.seed_inventory.items():
```

```
                text_surf = self.font.render(f'{item}: {amount}', False, 'Brown')
                bg_rect = pygame.Rect((screen_width / 2) - 150, top, 300, text_surf.get_height())
                pygame.draw.rect(self.display_surface, 'White', bg_rect, 0, 6)
                text_rect = text_surf.get_rect(center=(screen_width / 2, bg_rect.centery))
                self.display_surface.blit(text_surf, text_rect)
                top += text_surf.get_height() + 10
            for item, amount in self.player.seed_inventory.items():
                text_surf = self.font.render(f'{item}: {amount}', False, 'Brown')
                bg_rect = pygame.Rect((screen_width / 2) - 150, top, 300, text_surf.get_height())
                pygame.draw.rect(self.display_surface, 'White', bg_rect, 0, 6)
                text_rect = text_surf.get_rect(center=(screen_width / 2, bg_rect.centery))
                self.display_surface.blit(text_surf, text_rect)
                top += text_surf.get_height() + 10
        def input(self):
            keys = pygame.key.get_pressed()
            if keys[pygame.K_i] and not self.open_inventory:
                self.open_inventory()
            if keys[pygame.K_ESCAPE]:
                self.open_inventory()
        def update(self):
            self.input()
            if self.open_inventory:
                self.display_inventory()
```

(2.33)inventory.py class Inventory

Throughout the game, the player will notice that the game window will get darker over time, this is due to the daytime to nighttime transition. How this works is that when the game starts, the start color has an rgb value of [255,255,255] or white but because white is a shade, it doesn't change the color of the game. Throughout the game, the sky class will reduce each of the rgb values by -2 with the end color of [38,101,189].

sky.py > Sky > __init__

```
import pygame
from settings import *
from support import import_folder
from sprites import Generic
from random import randint, choice

class Sky:
    def __init__(self):
        self.display_surface = pygame.display.get_surface()
        self.full_surf = pygame.Surface((screen_width, screen_height))
        self.start_color = [255,255,255]
        self.end_color = [38,101,189]

    def display(self,dt):
        for index, value in enumerate(self.end_color):
            if self.start_color[index] > value:
                self.start_color[index] -= 2 *dt
        self.full_surf.fill(self.start_color)
        self.display_surface.blit(self.full_surf, (0,0), special_flags= pygame.BLEND_RGBA_MULT)
```

(2.34)sky.py class Sky

The transition class is used to set the speed of the changing rgb value, make the screen fade to black for the daytime transition and change the color to the default color of [255,255,255].

transition.py > Transition > play

```python
import pygame
from settings import *

class Transition:
    def __init__(self, reset, player):
        #setup
        self.display_surface = pygame.display.get_surface()
        self.reset = reset
        self.player = player
        #overlay image
        self.image = pygame.Surface((screen_width,screen_height))
        self.color = 255
        self.speed = -2

    def play(self):
        self.color += self.speed
        if self.color <= 0:
            self.speed *= -1
            self.color = 0
            self.reset()
        if self.color > 255:
            self.color = 255
            self.player.sleep = False
            self.speed = -2
        self.image.fill((self.color,self.color,self.color))
        self.display_surface.blit(self.image,(0,0), special_flags = pygame.BLEND_RGBA_MULT)
```

(2.35)transition.py class Transition

Lastly is the rainy weather, occasionally the game will rain and when it rains it will look something like this.

(2.36)rainy weather

This rain is done in the sky.py file using the class drop and rain. The drop class draws the rain droplets on the map while the rain class contains the rest of the rain aspect

```
21  class Drop(Generic):
22      def __init__(self, surf, pos, moving, groups, z):
23          #general setup
24          super().__init__(pos, surf, groups, z)
25          self.lifetime = randint(400,500)
26          self.start_time = pygame.time.get_ticks()
27          #moving
28          self.moving = moving
29          if self.moving:
30              self.pos = pygame.math.Vector2(self.rect.topleft)
31              self.direction = pygame.math.Vector2(-2,4)
32              self.speed = randint(200,250)
33      def update(self, dt):
34          if self.moving:
35              self.pos += self.direction * self.speed * dt
36              self.rect.topleft = (round(self.pos.x),round(self.pos.y))
37          if pygame.time.get_ticks() - self.start_time >= self.lifetime:
38              self.kill()
39  class Rain:
40      def __init__(self,all_sprites):
41          self.all_sprites = all_sprites
42          self.rain_drops = import_folder('../graphics/rain/drops/')
43          self.rain_floor = import_folder('../graphics/rain/floor/')
44          self.floor_w, self.floor_h = pygame.image.load('../graphics/world/ground.png').get_size()
45      def create_floor(self):
46          Drop(surf = choice(self.rain_floor),
47               pos = (randint(0,self.floor_w),randint(0,self.floor_h)),
48               moving = False,
49               groups = self.all_sprites,
50               z = layers['rain floor'])
51      def create_drops(self):
52          Drop(surf = choice(self.rain_drops),
53               pos = (randint(0,self.floor_w),randint(0,self.floor_h)),
54               moving = True,
55               groups = self.all_sprites,
56               z = layers['rain drops'])
57      def update(self):
58          self.create_floor()
59          self.create_drops()
```

(2.37)sky.py class Drop & Rain

The drop class uses a randint method imported from random for the lifetime or how long the droplets will appear on the map, in code the lifetime of each droplet is a random integer from 400 to 500 milliseconds, the direction of the droplet ensures that the droplet falls in an angle then the moving in the update function will animate the drops moving downwards towards the ground of the map. The rain class imports the drop image and the floor image, the floor is the animation when the rain droplets fall to the ground and become a puddle. What it does is make the drops

appear in the map and once the drops lifetime is over it turns the image into a random rain_floor from the floor folder.

III. Solution Design

A. Input and Outputs

1. Input

This game has multiple input possibilities, the possible input of the game includes keys such as up, down, left, & right to control movement, q to change the selected tools, w to change the selected seed, i to view inventory, esc to exit inventory and menu, spacebar to use tool and buy or sell items in menu, enter to interact with the trader and bed, left shift to plant seed. For this input example I will be displaying the nighttime to daytime transition that happens if the player interacts with the bed. In the input picture, we can see that the game window has changed colors significantly from the starting color which indicates that it is nighttime in the game.

(3.1.1)nighttime to daytime transition input

2. Output

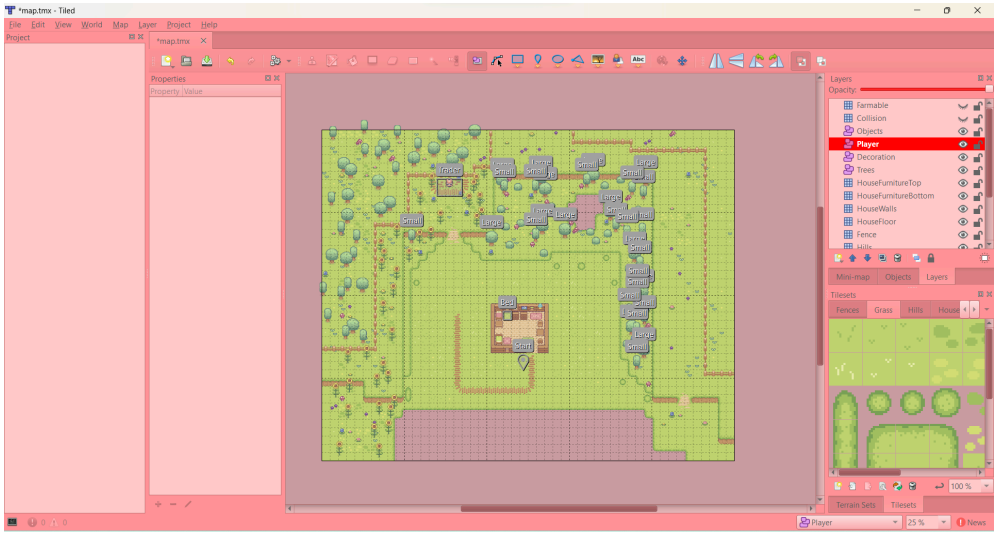
From the input possibilities, if the player presses up, down, left, or right the character will move in the corresponding direction, if the player presses q the game will go through a series of tools from hoe, axe, and water, if the player presses w the game will change the seed from corn to tomato, if the player presses i the game will freeze and display their inventory, if the player presses esc the player will exit the inventory and menu function, if the player presses spacebar they can use the selected tool or buy & sell items in menu, if the player presses enter they can interact with the trader if they're in front of the trader or sleep if they're beside the bed, if the

player presses left shift they can plant a seed in a soil patch. Following the input, the output given by the game if the player presses enter beside the bed, the game window would return to the original color which indicates that it's daytime in-game.

(3.1.2)nighttime to daytime transition output

B. Assets

The assets for the player, in-game objects, and map is an asset pack called Sprout Lands by Cup Nooble which could further be manipulated in terms of placement with tiled.

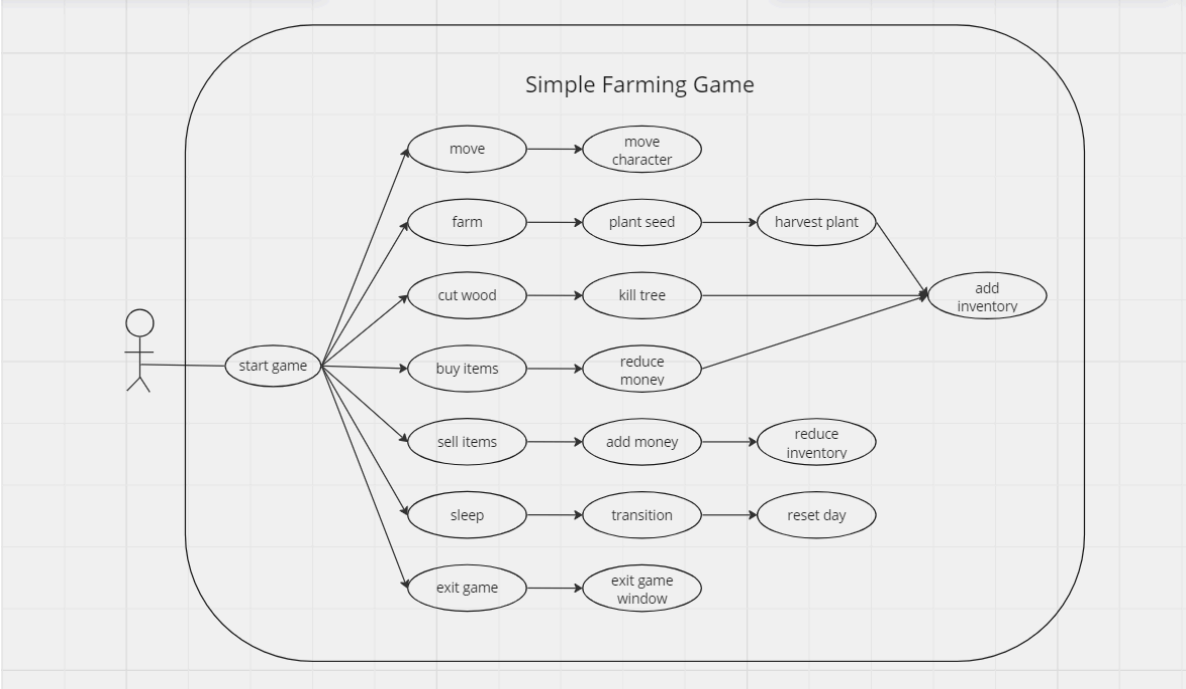


(3.2.1)map.tmx file of the game map

Sprout Lands Basic Pack by Cup Nooble

https://cupnooble.itch.io/sprout-lands-asset-pack

C. Use Case Diagram



(3.3.1)Use case diagram

D. Class diagram

Player
status = str
frame_index = int
speed = int
timers = dict
tools = list
tool_index = int
seed = list
seed_index = int
item_inventory = dict
seed_inventory = dict
money = int
sleep = bool
use_tool()
get_target_pos()
use_seed()
import_assets()
animate()
input()
get_status()
update_timers()
collisions()
move()
update()

CameraGroup
custom_draw()

Game
screen = Tuple((x,y))
run()

Level
shop_active = bool
inventory_active = bool
setup()
player_add()
toggle_shop()
open_inventory()
reset()
plant_collision()
run()

Menu
width = int
height = int
padding = int
options = list
sell_border = int
index = int
display_money()
setup()
input()
show_entry()
update()

Inventory
width = int
height = int
padding = int
display_money()
display_inventory()
input()
update()

Drop
lifetime = int
update()

Sky
display_surface = Tuple((x,y))
start_color = list
end_color = list
display()

Rain
create_floor()
create_drops()
update()

Timer
start_time = int
active = bool
activate()
deactivate()
update()

Transition
image = Tuple((x,y))
color = int
speed = int
play()

Generic
Particle(Generic)
duration = int
update()
Overlay
overlay_path = str
frame_path = str
display()

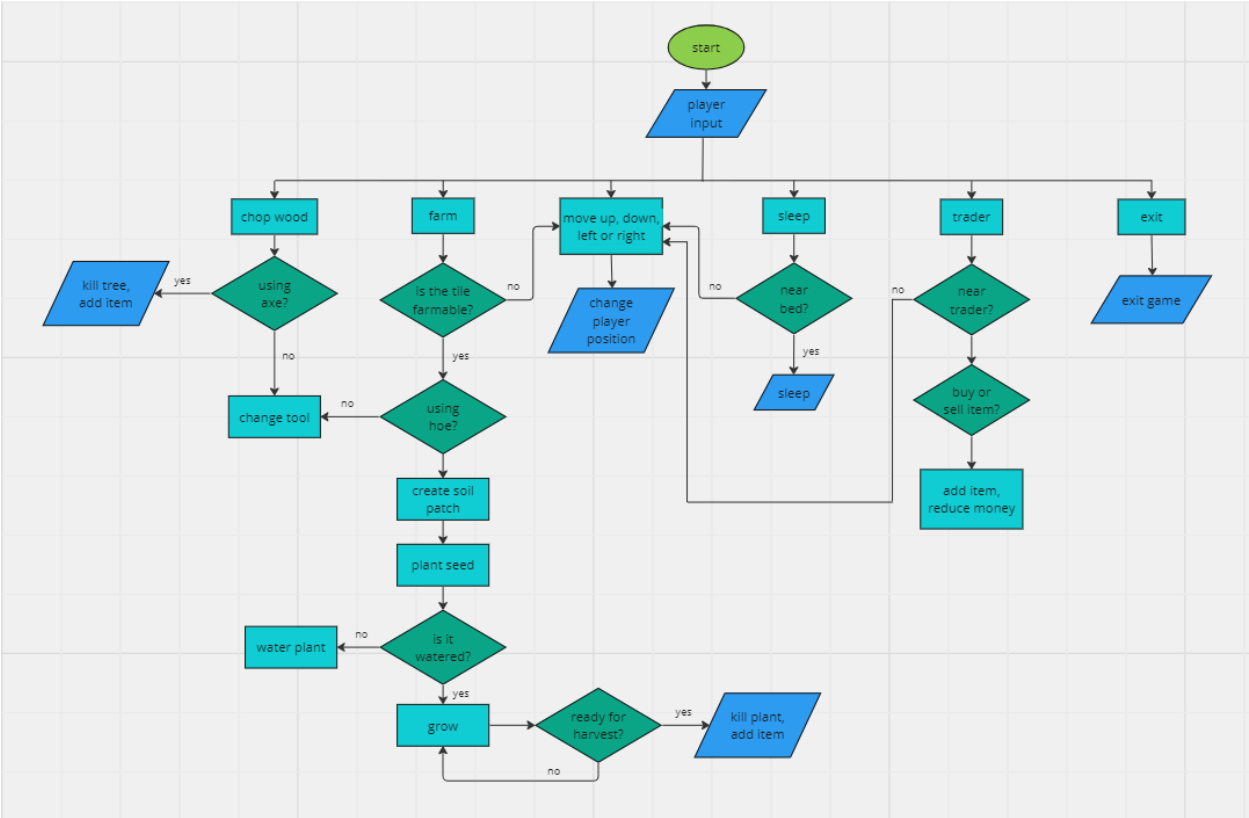
Interaction(Generic)
Tree(Generic)
health = int
alive = bool
damage()
check_death()
update()
regrow()
create_fruit()

Water(Generic)
frame_index = int
animate()
update()
SoilTile
z = int
WaterTile
z = int

WildFlower(Generic)
Plant
age = int
max_age = int
grow_speed = int
harvestable = bool
z = int
grow()

SoilLayer
create_soil_grid()
create_hit_rects()
get_hit()
water()
water_all()
remove_water()
check_watered()
plant_seed()
update_plants()
create_soil_tiles()

E. Flowchart



(3.5.1)Flowchart

IV. Data Structures

A. Lists & dictionaries

Most of the list and dictionaries used for this project is from a file named settings.py. This file contains a lot of dictionaries with each corresponding value to ensure the program works smoothly.

settings.py > ...

```python
overlay_positions = {
    'tool': (40, screen_height - 13),
    'seed': (70, screen_height - 3)}

player_tool_offset = {
    'left': Vector2(-50,40),
    'right': Vector2(50,40),
    'up': Vector2(0,-10),
    'down': Vector2(0,50)
}

layers = {
    'water': 0,
    'ground': 1,
    'soil': 2,
    'soil water': 3,
    'rain floor': 4,
    'house bottom': 5,
    'ground plant': 6,
    'main': 7,
    'house top': 8,
    'fruit': 9,
    'rain drops': 10
}

Apple_pos = {
    'Small': [(18,17), (30,37), (12,50),(30,45),(20,30),(30,10)],
    'Large': [(30,24),(60,65),(50,50),(16,40),(45,50),(42,70)]
}
```

```python
Grow_speed = {
    'corn': 1,
    'tomato': 0.7
}

sale_prices = {
    'wood': 4,
    'apple': 2,
    'corn': 10,
    'tomato': 20
}

purchase_prices = {
    'corn': 4,
    'tomato': 5
}
```

(4.1.1)settings.py

Other than settings.py, there's another file I used in my project that turns a folder filled with images into a dictionary or a list of surfaces. This file named support.py helps me automatically import a lot of images into the program without having to import them all one by one manually.

support.py > import_folder_dict

```python
from os import walk
import os.path
import pygame

def import_folder(path):
    # Turn the contents of the folder into a list of surfaces
    surface_list = []

    for _, __, img_files in walk(path):
        for image in img_files:
            full_path = path + '/' + image

            # Load image and convert to surface
            image_surf = pygame.image.load(full_path).convert_alpha()
            surface_list.append(image_surf)

    return surface_list

def import_folder_dict(path):
    surface_dict = {}
    for _, __, img_files in walk(path):
        for image in img_files:
            full_path = path + '/' + image

            # Load image and convert to surface
            image_surf = pygame.image.load(full_path).convert_alpha()
            surface_dict[image.split('.')[0]] = image_surf
    return surface_dict
```

(4.1.2)support.py

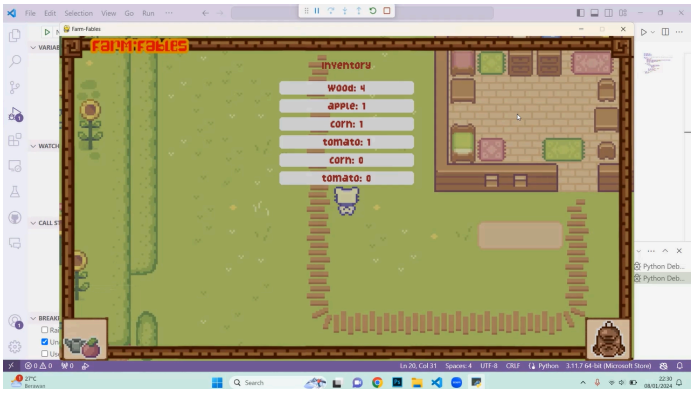
B. Classes

As I'm creating a game for my project, I used a lot of class implementation. Each class has a different function in the game, some are more simple and some are more complicated. One of the main components of my game is the Player class. This class keeps track of how many items are in the player inventory, the player inputs, and it also sets the game logic of the things the player can interact with.

```
player.py > Player > input
 1   import pygame
 2   from settings import *
 3   from support import *
 4   from timer import Timer
 5
 6   class Player(pygame.sprite.Sprite):
 7       def __init__(self, pos, group, collision_sprites, tree_sprites, interaction, soil_layer, ···
65
66       def use_tool(self):···
76
77       def get_target_pos(self):···
79
80       def use_seed(self):···
84
85       #make sure to import the right asset
86       def import_assets(self):···
96
97       #make the image animate
98       def animate(self,dt):···
104
105
106      def input(self):···
179
180
181      def get_status(self):···
189      #make sure to not loop the tool use forever
190
191      def update_timers(self):···
194
195      def collision(self, direction):···
```

(4.2.1)player.py class Player

C. Strings



(4.3.1)Inventory

String implementation is an important aspect of my game, in this case strings are used in a dictionary, specifically the item_inventory and seed_inventory. These strings are essential for my project as they help the player know what items they have inside their inventory.

V. Program Walkthrough

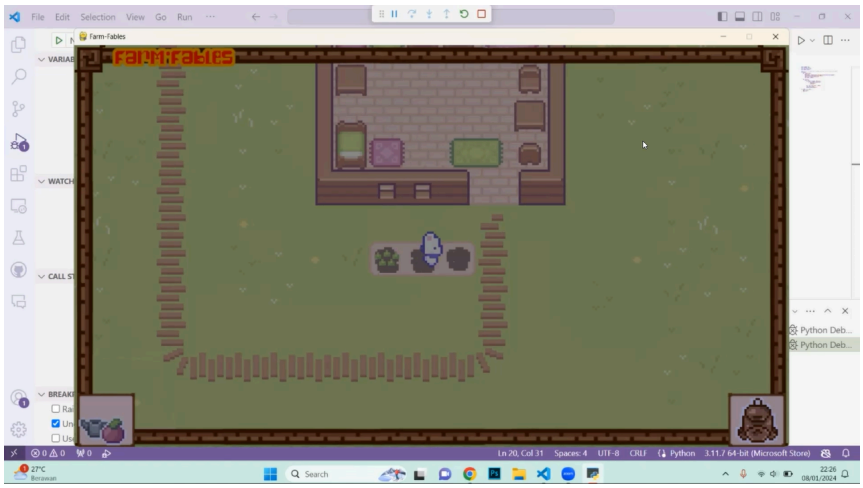
A. Player movement

The player could use the up, down, left, right keys to make the character move in a certain direction. When walking or idle, the player has a specific animation associated with each movement. The player could also use a variety of tools such as hoe, axe, and a watering can by pressing the q key and pressing spacebar to use the tools. The tools enable the player to farm or collect wood and apples.

(5.1)tool use(axe)

B. Farming

The player could farm by creating a soil patch with a hoe in the farmable area, planting a seed, then watering it with the watering can so that the seed could grow. The player could plant either corn or tomato in their field. The player can switch the seed by pressing the w key and pressing left shift to use the seed. When the player goes to sleep, the seed will grow according to each of their growth speeds. The next day, the player must water their plants again if they wish to see them grow. This process will go on for a couple of days until the plants stop growing and can be harvested.



(5.2)Farming

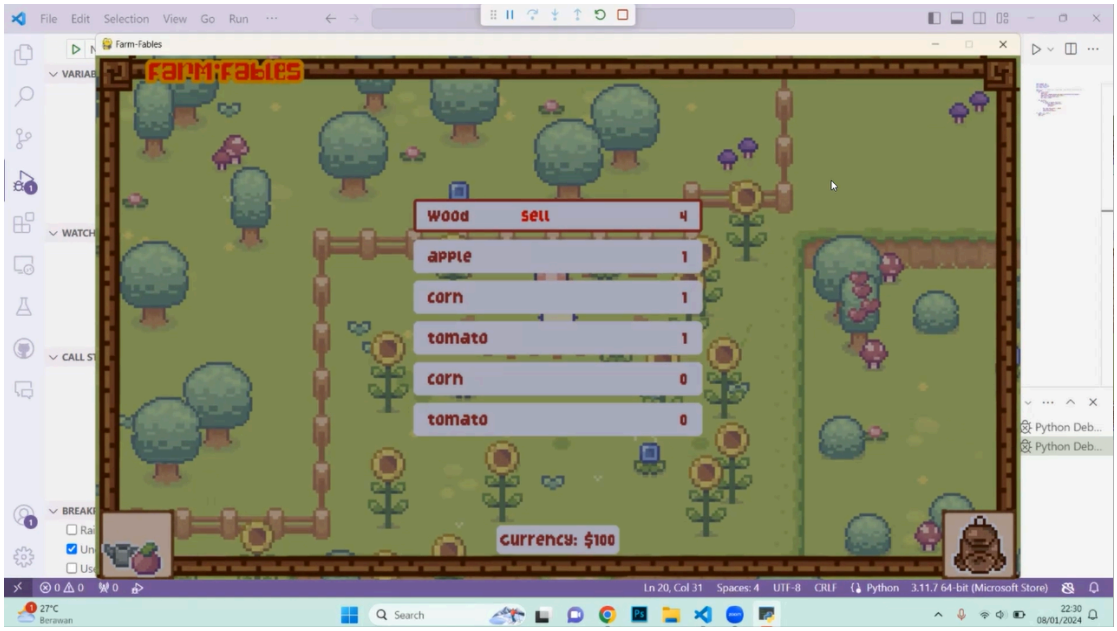
C. Collecting wood

By using the axe tool in a nearby tree, the player could collect wood by chopping down trees. Each tree has 5 health bars so once the player has hit the tree 5 times using their axe, they're able to obtain wood. Each tree has a random amount of apples growing on them, with each strike the player aims for the tree, a random apple falls and then added to their inventory. All of the tree stumps will grow back on the next day after the player sleeps.

(5.3)collecting wood

D. Trader

The trader is an area in the corner of the map where the player could interact with to sell or buy items. The items the player has collected such as corn, tomatoes, wood, and apples could then be sold off to the trader and the player will earn some money with each item sold. Once the player has run out of seeds, they can also buy more seed in the trader by using the money they have earned by selling their items. The player could press enter to interact with the trader and open the menu function, then they can choose from a selection of items by using the up & down key and pressing the spacebar if they want to buy or sell something.



(5.4)Trader

E. Daytime to nighttime transition

With each passing moment in the game, the game window will get darker. This is due to the daytime to nighttime transition function. After the player has started the game, the game window would gradually get darker. The player could resume their activities as they normally would but the game window would be dark, indicating that it's nighttime in the game. If the game window has gotten to the nighttime transition, the player could sleep by standing beside the bed and pressing enter.

(5.5)Daytime

(5.6)Nighttime

F. Weather

Every once in a while, it will rain. There will be random droplets of water and puddles all throughout the screen randomly. This rain could help if the player is farming and waiting for the plant to harvest as the rain will water all of the soil patches so that the player doesn't need to individually water them anymore.

(5.7)rainy weather

VI. Evaluation & Reflection

Evaluation

I have encountered many errors and obstacles throughout his intricate project, since this game consists of many different aspects. The first error I encountered was with the player animations and the movement of the player, I simply can't upload the player into the game window I created. Turns out I forgot to add a line in the update function but then the player's movement seemed slow and laggy despite already adding a delta time for the movements. After hours of rechecking the code I just created, I realized it's because of the code indentation. Later on when I finished importing the map and trees, I encountered another problem with creating the apples. I was able to fix this by replacing a line from "from sprites import *" to "from sprites import Generic, Water, WildFlower, Tree, Interaction, Particle." Even after finishing the project I still encountered problems with the apple from using Vscode, I'm able to come to this conclusion

because sometimes when I run the code, the apples don't appear and when I cut down a tree it produces an error but when I refresh without changing the code, the apples appear and there's no longer an error.

Reflection

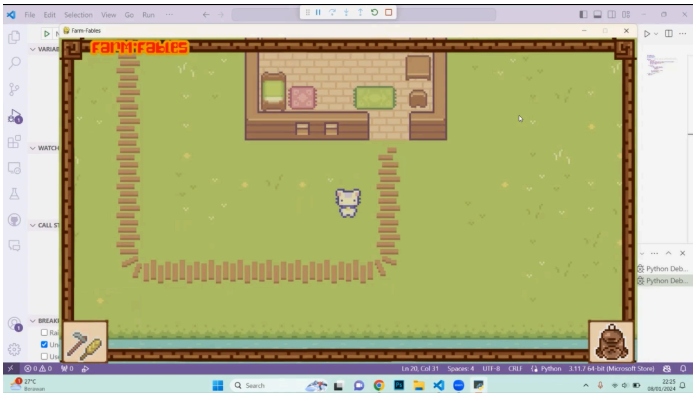
I've had a couple different ideas for the final project that I scrapped since it seemed too simple or has too many assets to draw, I settled on this game simply because I love the aesthetics of the game, it wasn't a common game project, and it has a detailed tutorial to help me understand. At first I watched a detailed introduction video to better understand pygame by making a simple offline chrome dino game in order to grasp the understanding of game logic and pygame functions only then I'm able to move on to the farming game as it's quite an advanced project that requires prior python and pygame understanding. By watching the basic pygame tutorial, I'm able to better understand the simple aspects needed in a game such as windows, rectangles, simple animations, etc. I'm very grateful that I was able to stumble across this simple farming game tutorial as the contents of the tutorial are detailed and easy to understand.

Working on this project is certainly a challenge for me since it's more advanced, contains multiple files, classes, and functions. There are multiple classes and functions that I struggle with especially in the beginning of the process. The function that I have a really hard time with is the animate function and the player status. As these functions are created far in the beginning, I don't have nearly as much understanding of this project as I now have which is why I struggled. I tried debugging with several resources online, using different coding applications but in the end I'm able to identify and fix the problem, thankfully it's not hard to do as I've just missed one update line in the code. There are a couple more minor problems I encountered throughout the project and thankfully able to solve during this project. The last challenging problem I encountered was when I wanted to add a duck to my game, the duck was supposed to be a little companion for the player, to follow them around when they're moving. But with the upcoming presentation and the many errors the duck has, I have to make the difficult decision to not include the duck in the final game.

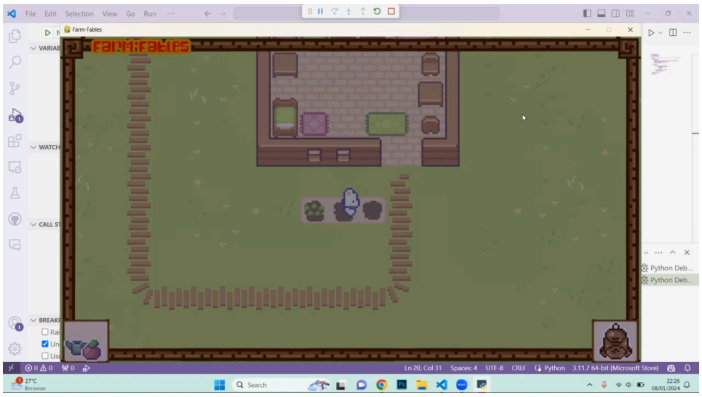
This project has not been the easiest so far but I'm grateful that I chose this project to do for my final project. This project has helped me understand more about coding and made me familiar with pygame. Not only did I learn a lot by completing this program, I'm also able to enjoy what I'm making, with each problem I manage to debug I feel a sense of satisfaction &

pride for being able to understand what I'm doing and not only following blindly on the instructions given.

VII. Working Program Evidence

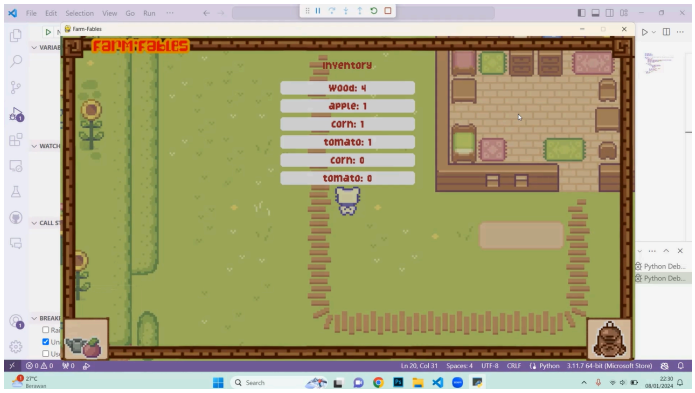


(7.1)Player Starting Point

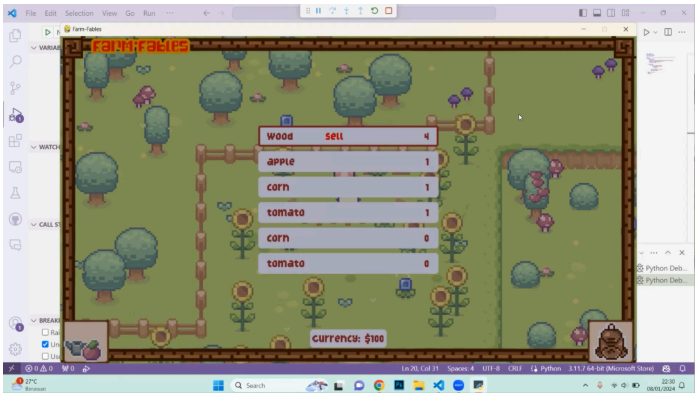


(7.2)Farming Mechanics

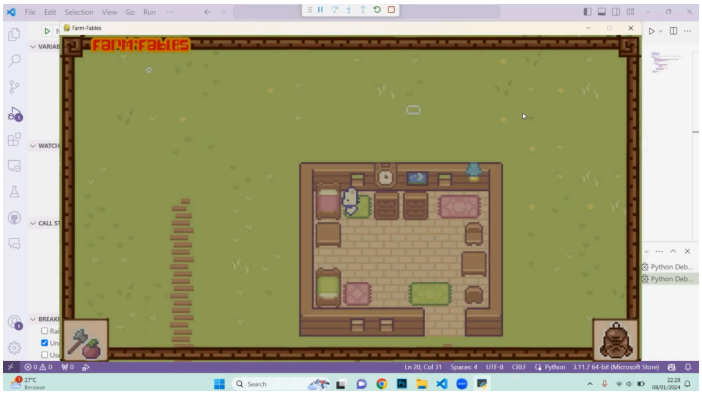
(7.3)Chopping wood



(7.4)Player Inventory



(7.5)Sell or Buy Function



(7.6)Raining

VIII. Resources & Assets

Github repo - https://github.com/smoliguana/Farm-Fables/tree/master

Sprout Lands free assets - https://cupnooble.itch.io/sprout-lands-asset-pack

Demo video - https://drive.google.com/drive/folders/1-e_yhGAu1izlbVLKt3IfbAwFiucFKJse?usp=sharing

IX. References

Basic pygame explanation - https://youtu.be/AY9MnQ4x3zk?si=MwWFXPi6DvXp_Svr

Tiled tutorial - https://youtu.be/N6xqCwblyiw?si=PjVsQo9_labrEJ3G

Simple farming game tutorial - https://youtu.be/T4IX36sP_0c?si=LYtwnAAcIabjo7nx