Final Report
Algorithm and Programming

Lecturer:

Jude Joseph Lamug Martinez, MCS

Made by:

Ariel Prandi Darmawidjaja - 2702337584

BINUS UNIVERSITY INTERNATIONAL

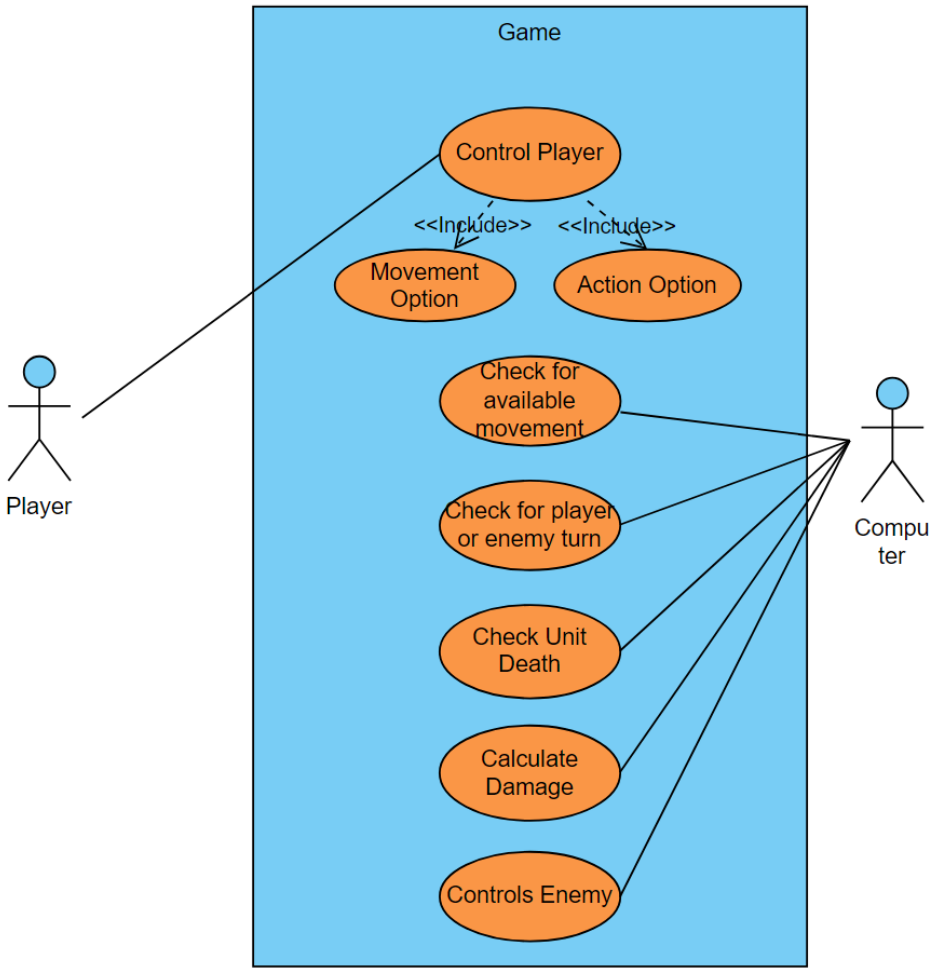
JAKARTA

2023

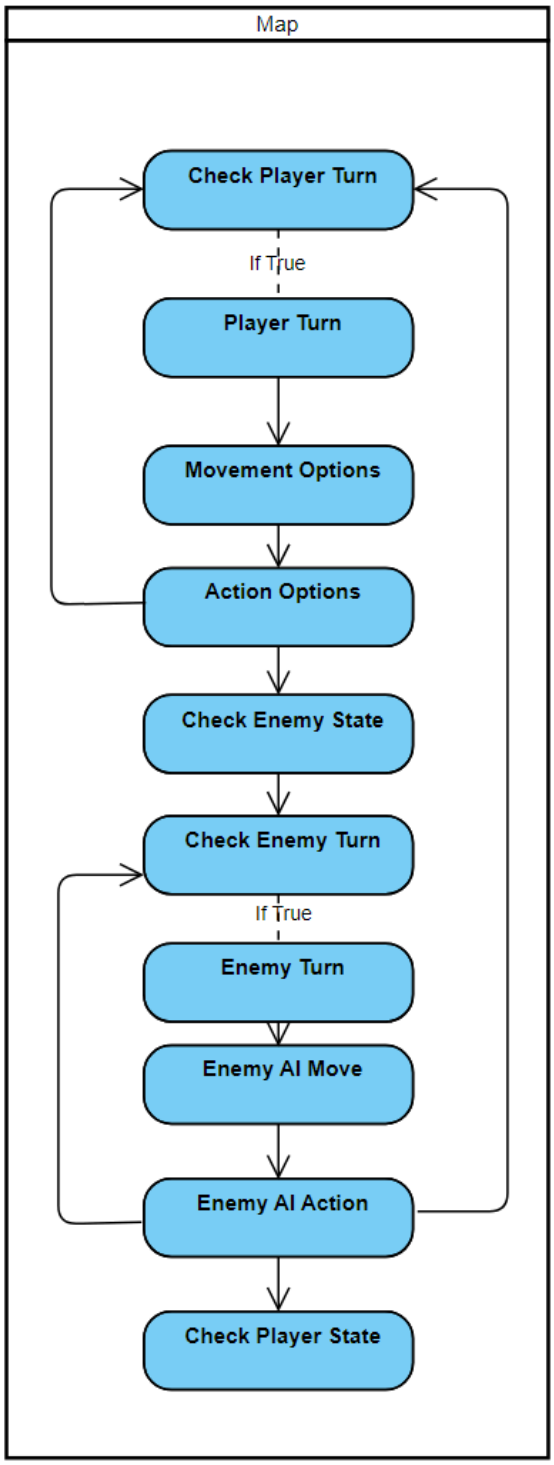
I. Brief Description

My final project is a tactical turn-based rpg based on the Game Boy Advance game Fire Emblem. The game was made with pygame. The game starts off directly as the program is run. The objective of the game is to defeat the enemies unit (ghost sprites) with your own controllable units (non-ghost sprites). The player units can be selected by hovering over them and pressing left click. They are able to move 5 cells along a grid, with W-S-A-D keys and perform an action. Once all player units have performed an action, the enemy units can start moving and performing actions. Each turn plays out this way until either all enemies or player units have been defeated.

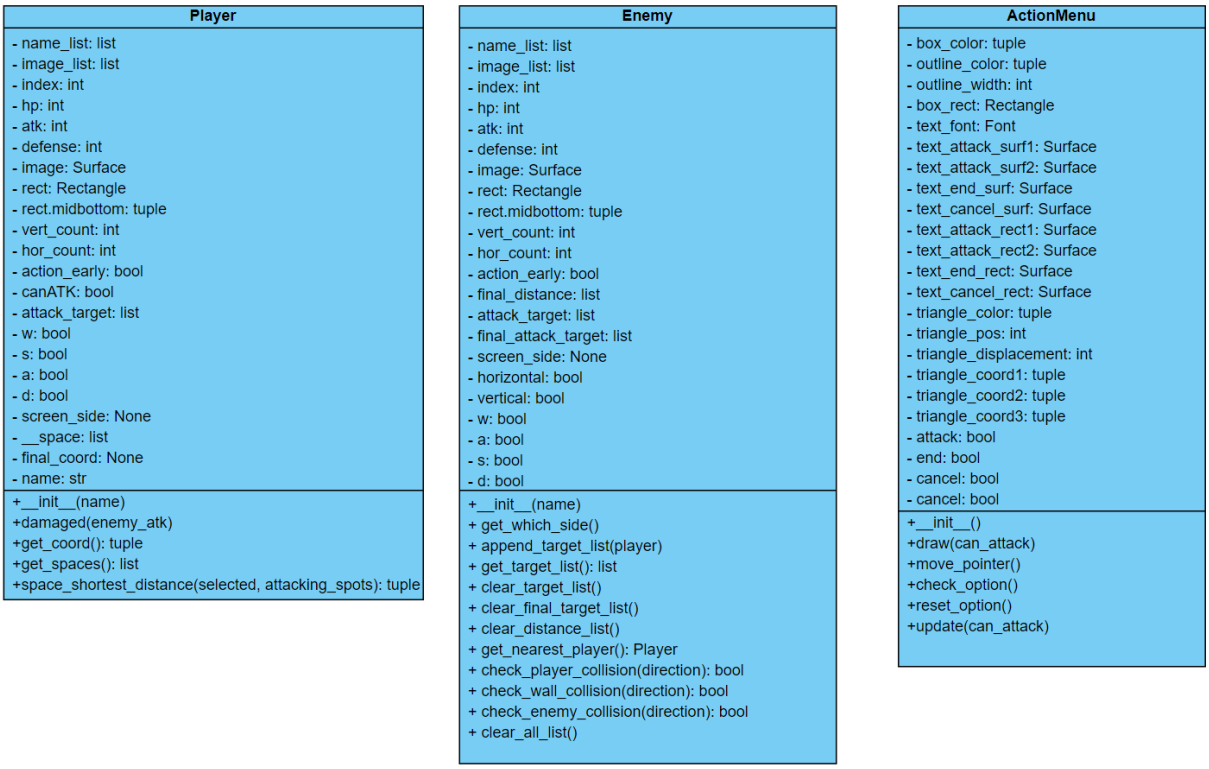
II. Use-case Diagram



III. Activity Diagram



IV. Class Diagram



V. Modules

A. Pygame Module

The pygame module was used in the construction of the game. Allows for the easy display of images by the in-module Surface class, handling of collision by the in-module Rectangle class and handling input.

B. sys Module

The sys module was only used to import the exit() function so that the game window can be closed

C. grid2 Module

The grid2 module contains the matrix for the environment matrix (matrix for collision) and the grid matrix (matrix containing grid coordinates)

D. unit_stats

The unit_stats module contains dictionary containing stats of player units and enemy units.

E. math

The math module was used for the sqrt function.

VI. Essential Algorithms

A. Player Collisions Algorithms:

I utilized three functions for the player's collision with the map environment, enemy units, and other player units. First, the collision with the map environment utilizes the grid player's position within the grid matrix and then its corresponding position in the environment matrix. If the element in that position in the environment matrix is 1, it means the space is traversable, otherwise it isn't traversable. Second, collision with enemy units involves a function which accepts two parameters, that is the selected player and the event key (direction of movement). The program iterates through the enemies Sprite group, and for each event key, it checks for collision between the rect of selected player and the center rect of each enemy. If there is collision, the player rect is pushed in the opposite direction of its initial movement. Similarly, the collision between player units also makes use of a function that accepts two parameters of player and event key (direction of movement). The program iterates through the players Sprite group, and for each event key, it checks for collision between the rect of the selected player and the center rect of each player that isn't the selected player. For example, if there is an input of W key, the player y rect increases in the negative direction, if there is collision with another player unit's center rect, the player y rect increases in the positive direction by the same magnitude to give the illusion of no movement.

B. Enemy Collisions Algorithm:

I utilized three class methods for the enemy's collision with map objects, enemy units,

and other player units. Unlike the player's collision with the map environment, the enemy's collision with the map environment did not reference the grid matrix. This is the case because at the time of creating the algorithm for the player's collision, I did not yet realize that I could easily get the position of the unit (in terms of grid coordinate) by just getting the center rect of the unit and floor division with the length of each cell. So, collision between enemy and map environment is a method accepting the direction (of movement) parameter utlising an if statement for the direction of enemy movement, and it checks if the position one cell away (relative to the direction, e.g., if moving right, checks one cell to the right) from the enemy is a 1 or a 0 in the environment matrix. Code snippet:

get_coord() method gets the position of the player in terms of grid coordinates.

```
# references the env_matrix whose value of [row][col] equals the coordinates of the grid (x, y)
def check_wall_collision(self, direction):
    if direction == 'down':
        if env_matrix[self.get_coord()[1] + 1][self.get_coord()[0]] == 0:
            return False
        else:
            return True
    elif direction == 'up':
        if env_matrix[self.get_coord()[1] - 1][self.get_coord()[0]] == 0:
            return False
        else:
            return True
    elif direction == 'left':
        if env_matrix[self.get_coord()[1]][self.get_coord()[0] - 1] == 0:
            return False
        else:
            return True
    elif direction == 'right':
        if env_matrix[self.get_coord()[1]][self.get_coord()[0] + 1] == 0:
            return False
        else:
            return True
```

Next, the collision for enemy between enemy and enemy between player have the same logic, however they are different methods. The method takes the parameter direction and has blocks of if and elifs for each direction. Similar to the enemy-wall method, it checks for the space one cell away from the unit (relative to direction of movement). However, the difference is instead of checking with the environment matrix the program iterates through the enemies and players Sprite group, respectively, and gets the grid coordinate of each sprite object with the get_coord()

method, and sees if one cell away from the unit is equal to the coordinate of another sprite object. If it is equal, which means there is an enemy/player occupying the space, returns False, otherwise returns True.

C. Player Input Algorithm:

Nested within the event loop, and nested within the for loop that iterates through the players Sprite group. First, to select the player, the program checks if the player has performed an action before by seeing if the player name is in the the list of player names (list for players that have performed an action) and it also checks if the game is in players turn or enemy's turn. If both conditions are fulfilled, the way a specific player is selected is by matching the player index attribute and taking the mouse position and matching it to the index of the player position in the player positions list. The index of the position of the player in the list corresponds to the index of the player. So, if they match, the program knows which specific player is selected. Next, the program takes input in the form of pygame.Key_w/s/a/d for player movements.

D. Checking Available Player Moves Algorithm:

This algorithm checks how many more cells the player can move from the original position. Does this by adding a vertical count to each vertical movements and horizontal count to each horizontal movement. Vertical count increases by 1 for every w key pressed, and -1 for every s key pressed. Horizontal count increases by 1 for each d key pressed, and -1 for each a key pressed. If the sum of the absolute values of vertical count and horizontal count reaches 5, the input keys will stop affecting player movement, and the action menu is brought up.

E. Targetable Enemy Algorithm:

Iterates through players Sprite group, and then iterates through enemies Sprite group, then checks for each enemy if enemy is in the player class attack_target attribute. If not present, the program checks for collision between the player rect and midtop, midleft, midbottom, and midright point of the enemy. If collision midtop is True, .s attribute for player and enemy are also True, if midleft is True, .right attribute for player and enemy are also True, if midbottom is True,

.w attribute is also True. When the action menu is blitted, for each of those attributes which are True, their respective keys are blitted above the heads. Then, if the attack option is selected the user can press the keys that are blitted to attack the enemy.

F. Enemy Movement Algorithm:

Enemy will move if the sum of the absolute vertical and horizontal count is less than 5, the enemies are able to move. The enemies and players each have a 'screen_side' attribute, which are strings 'right' or 'left'. Enemies on the 'left' can only target players on the 'left', and enemies on the 'right' can only target players on the 'right'. First, the enemies target the player closest to them, then they see if the player is within their detection radius of 5 sqrt2. If both conditions are not fulfilled, the enemy will not move. If the conditions are fulfilled, the enemy will target the player closest to them by checking for the open cells adjacent to the player. To that, I create a player class method to check for the open spaces. First, the get_spaces() method will append the coordinates of all adjacent cells (top, left, bottom, right) into a list. Then, to check if there is a player or enemy occupying the open spaces, the program iterates through the players Sprite group and enemies Sprite group and checks if the position of any of the sprites are in that list of positions; remove position from the list if they match. Next, it also checks if the map environment is also in the open space by iterating through the list of open spaces and sees if the corresponding position in the environment matrix is a 0 or a 1; remove position from the list if 0.

Once the occupied positions have been removed, the euclidean distance between the enemy and the last open spaces is calculated, and the enemy will choose to move towards the spot with the shortest distance.

The code that handles vertical movement are separated into four if blocks handling the four directions (up, down, left, right). Essentially, the program checks whether the enemy is above, below, to the left, or to the right of the target space. For example, if above, they will move down until a collision occurs or until the same height or until 5 spaces have been moved or until reaches the target space, if collision occurs it will check for the other axis. It will continue doing this in an intermittent fashion until 5 spaces have been moved or the enemy has reached the

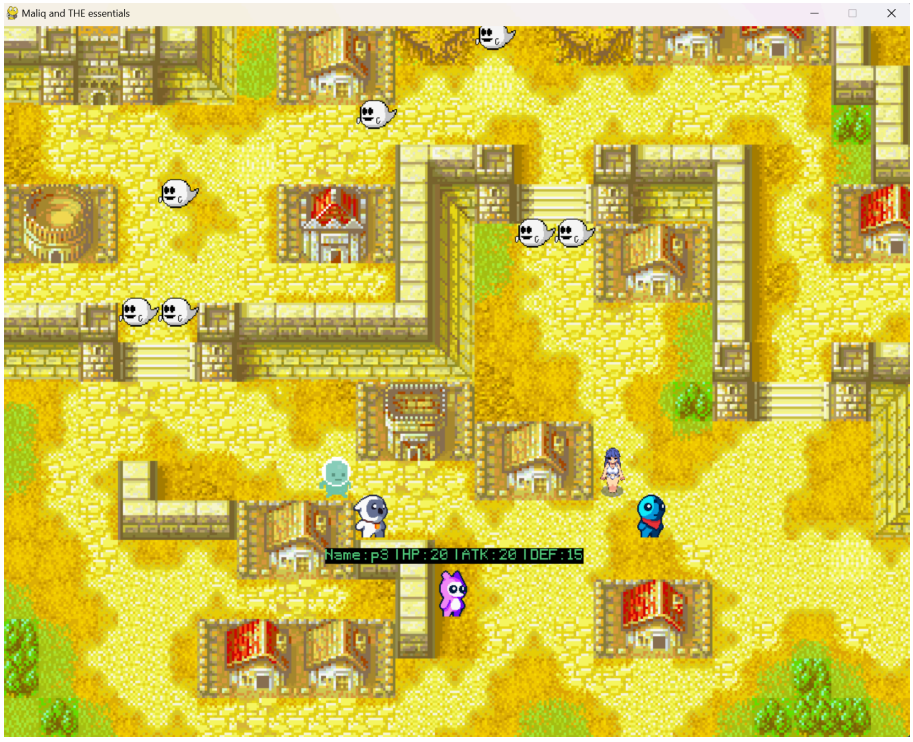
target space. Snippets of the code:

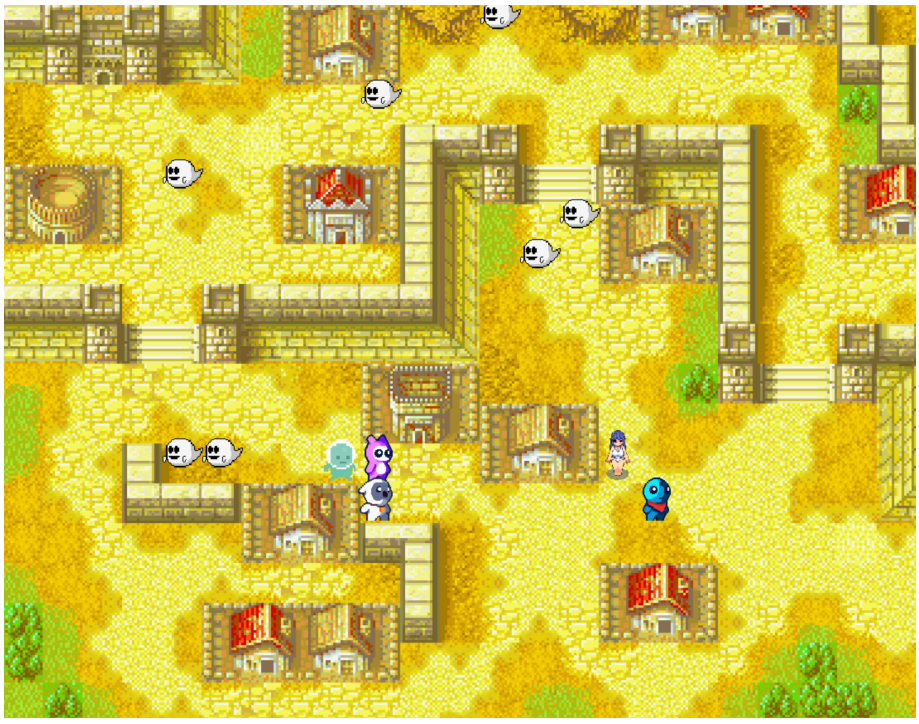
```
# find the smallest euclidean distance and target that! ! ! ! ! !
if ((abs(selected_enemy.vert_count) + abs(selected_enemy.hor_count)) < DISTANCE_THRESHOLD and
        enemy_coord not in attacking_spots and final_spot is not None):
    # moving down
    if (enemy_coord[1] < final_spot[1] and selected_enemy.check_player_collision(nearest_player, 'down')
            and selected_enemy.check_enemy_collision(selected_enemy, 'down')
            and selected_enemy.check_wall_collision('down')):
        pygame.time.delay(200)
        selected_enemy.rect.y += cell_total
        selected_enemy.vert_count -= 1

    # moving right
    elif (enemy_coord[0] < final_spot[0] and selected_enemy.check_player_collision(nearest_player,'right')
          and selected_enemy.check_enemy_collision(selected_enemy, 'right')
            and selected_enemy.check_wall_collision('right')):
        pygame.time.delay(200)
        selected_enemy.rect.x += cell_total
        selected_enemy.hor_count += 1

    # moving up
    elif (enemy_coord[1] > final_spot[1] and selected_enemy.check_player_collision(nearest_player,'up')
            and selected_enemy.check_enemy_collision(selected_enemy, 'up')
            and selected_enemy.check_wall_collision('up')):
        pygame.time.delay(200)
        selected_enemy.rect.y -= cell_total
        selected_enemy.vert_count += 1
    # moving left
    elif (enemy_coord[0] > final_spot[0] and selected_enemy.check_player_collision(nearest_player,'left')
          and selected_enemy.check_enemy_collision(selected_enemy, 'left')
          and selected_enemy.check_wall_collision('left')):
        pygame.time.delay(200)
        selected_enemy.rect.x -= cell_total
        selected_enemy.hor_count -= 1
```

VII. Evidence of Working Program


Maliq and THE essentials
Attack
End
Cancel


Maliq and THE essentials
Attack
End
Cancel


Maliq and THE essentials
Name:p3 |HP:20 |ATK:20 |DEF:15

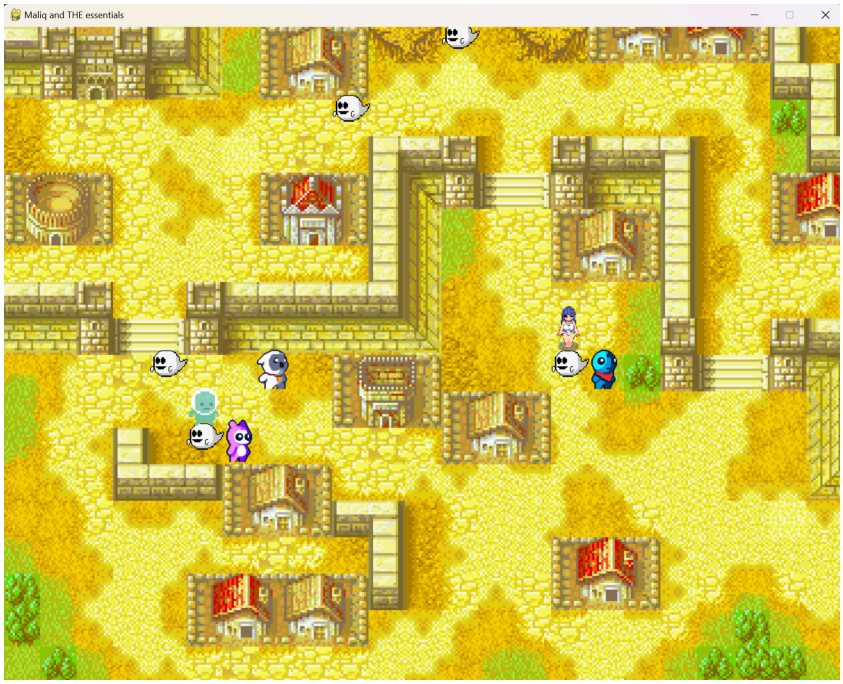


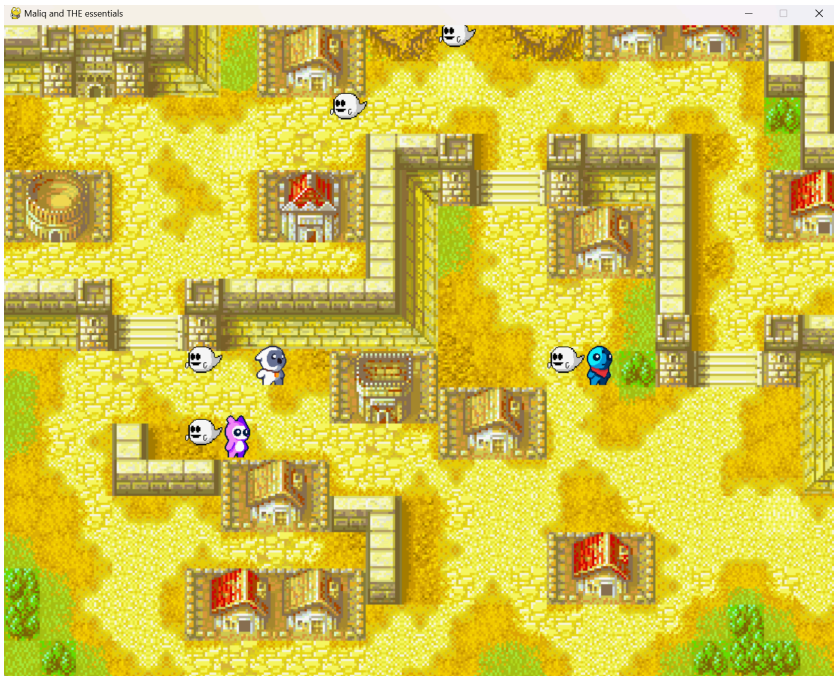

Maliq and THE essentials
Attack
End
Cancel


Maliq and THE essentials
Name:e2|HP:15|ATK:20|DEF:10


Maliq and THE essentials
Attack
End
Cancel


Maliq and THE essentials


Maliq and THE essentials


Maliq and THE essentials

VIII. Reflection

First and foremost, I learned that my time management skills were lacking. I initially thought that 2 weeks were going to be enough for this project, in reality it took me a month and many sleepless nights. However, I am grateful for this project because I feel like I have also gained knowledge and experience and that my logic has improved ever since starting the project. This was especially noticeable because when I look back at my code from the start of last month, I can now find ways to improve it and make it more efficient. This shows that I have grown as a python programmer.