AOOP Final Project: Bomb Man Game

Jun-Wei,Li
Department of Electonics and Electrical Engineering
National Yang Ming Chiao Tung University
Hsinchu City, ROC
peter921123.en11@nycu.edu.tw

***Abstract*—This is a game that players have to place bomb to blow up their enemy and avoid being blown up.**

I. Introduction

In this game, you control the character with the w, a, s, and d keys and place bombs with the space bar key. Three types of items may appear when obstacles - boxes are destroyed. They can either increase the player's speed, the maximum number of bombs the player can place, or the attack range of the player's bomb.

II. OOP Design

A. Class Diagram

Three are major parts of class diagram.

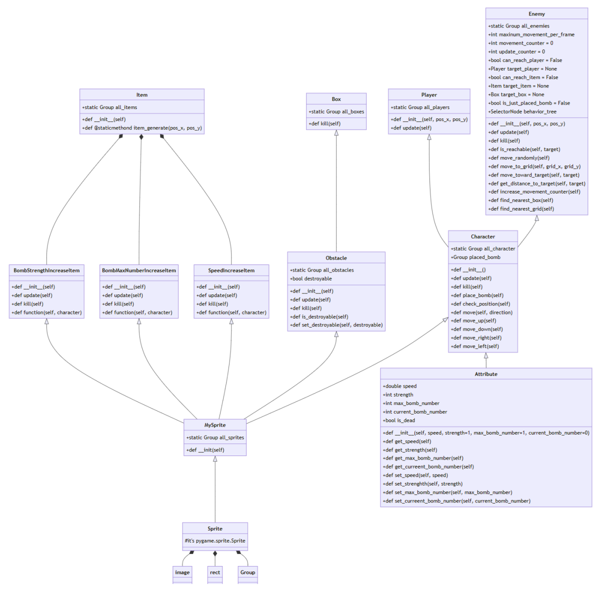
Item
+static Group all_items
+def __init__(self)
+def @staticmethond item_generate(pos_x, pos_y)
Box
+static Group all_boxes
+def kill(self)
Player
+static Group all_players
+def __init__(self, pos_x, pos_y)
+def update(self)
Enemy
+static Group all_enemies
+int maxinum_movement_per_frame
+int movement_counter = 0
+int update_counter = 0
+bool can_reach_player = False
+Player target_player = None
+bool can_reach_item = False
+Item target_item = None
+Box target_box = None
+bool is_just_placed_bomb = False
+SelectorNode behavior_tree
+def __init__(self, pos_x, pos_y)
+def update(self)
+def kill(self)
+def is_reachable(self, target)
+def move_randomly(self)
+def move_to_grid(self, grid_x, grid_y)
+def move_toward_target(self, target)
+def get_distance_to_target(self, target)
+def increase_movement_counter(self)
+def find_nearest_box(self)
+def find_nearest_grid(self)
BombStrengthIncreaseItem
+def __init__(self)
+def update(self)
+def kill(self)
+def function(self, character)
BombMaxNumberIncreaseItem
+def __init__(self)
+def update(self)
+def kill(self)
+def function(self, character)
SpeedIncreaseItem
+def __init__(self)
+def update(self)
+def kill(self)
+def function(self, character)
Obstacle
+static Group all_obstacles
+bool destroyable
+def __init__(self)
+def update(self)
+def kill(self)
+def is_destroyable(self)
+def set_destroyable(self, destroyable)
Character
+static Group all_character
+Group placed_bomb
+def __init__()
+def update(self)
+def kill(self)
+def place_bomb(self)
+def check_position(self)
+def move(self, direction)
+def move_up(self)
+def move_down(self)
+def move_right(self)
+def move_left(self)
Attribute
+double speed
+int strength
+int max_bomb_number
+int current_bomb_number
+bool is_dead
+def __init__(self, speed, strength=1, max_bomb_number=1, current_bomb_number=0)
+def get_speed(self)
+def get_strength(self)
+def get_max_bomb_number(self)
+def get_curreent_bomb_number(self)
+def set_speed(self, speed)
+def set_strenghth(self, strength)
+def set_max_bomb_number(self, max_bomb_number)
+def set_curreent_bomb_number(self, current_bomb_number)
MySprite
+static Group all_sprites
+def __init(self)
Sprite
#it's pygame.sprite.Sprite
image
rect
Group

Fig. 1. In-Game Component Class Diagram

B. Classes

Below are the introduction of the classes and their major function in the game.

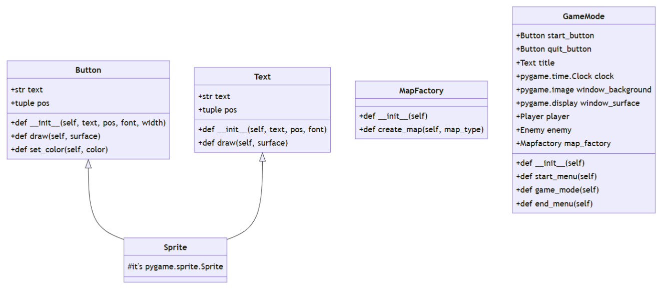

Fig. 2. Game-Setting Class Diagra

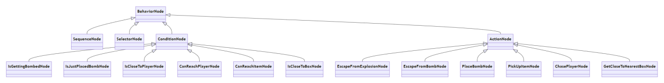

Fig. 3. Behavior Tree Class Diagram

- Class MySprite:
  - The MySprite class is a subclass of pygame.sprite.Sprite and serves as a representation of a sprite in a Pygame application.
  - The class manages all instances of MySprite through the all_sprites attribute, which is a pygame.sprite.Group. Upon the creation of a new MySprite instance, it is automatically added to the all_sprites group.
  - The primary purpose of this class is to act as a base class for creating sprites within a Pygame application. It facilitates the convenient management and updating of all sprites within a specified group.
- Class Item:
  - The Item class represents a container class that defines three different types of items in the game: BombStrengthIncreaseItem, SpeedIncreaseItem, and MaxBombNumberIncreaseItem.
  - The Item class serves as a container and has a static attribute all_items to keep track of all item instances created.
  - The Item class also contains a static method item_generate() that generates a random item based on probabilities and adds it to the all_items group.

- Class Obstacles:
  - The Obstacle class represents obstacles in a game environment. It inherits from mysprite.MySprite and has a static attribute all_obstacles to keep track of all instances of the Obstacle class created. On top of that, it has a boolean attribute is_destroyable that decides whether the object can be destroyed by the bombs.
  - The Obstacle class overrides the update and kill method from the parent class but does not contain any specific implementation, it is implemented in its derived classes.
- Class Box:
  - The Box class represents a specific type of obstacle in the game, inheriting from the obstacle.Obstacle class. The class has a static attribute called all_boxes to keep track of all instances of the Box class created.
  - The attribute is_destroyable is set True. So the instances of the Box class can be destroyed by the bombs.
  - Overriding the kill method from the parent class. It calls the item_generate() method of the Item class from the items module with the current position of the box before calling the parent kill method. This method is responsible for generating an item and adding it to the game when a box is destroyed.
- Class Attribute
  - The Attribute class is a utility class that encapsulates various attributes associated with the game entities. It's data members include default values for speed, strength, max_bomb_number, current_bomb_number, and is_dead. On top of that, the Attribute class includes the setter and the getter methods of attributes mentioned above.
- Class Character
  - The Character class acts as a comprehensive representation of a game character, inheriting attributes from both the Attribute and MySprite classes. The class has a static attribute called all_characters and placed_bombs to keep track of all instances of the Character class created and bombs placed by the certain player.
  - The update method is pivotal for maintaining the character's state. It checks and updates the character's position and the count of placed bombs. The check_position method ensures the character remains within the game window boundaries.
  - For movement control, the class implements the move method, allowing the character to move in a specified direction.
  - The place_bomb method control bomb placement, checking if the character has reached the maximum limit of placed bombs, incrementing the count, and creating a new bomb instance at the character's position.
- Class Player
  - The Player class is a specialized form of the Character class, inheriting its attributes and behaviors, representing a player character in the game. The class has a static attribute called all_characters to keep track of all instances of the Characters class created.
- Class Enemy
  - The Enemy class is a specialized form of the Character class, inheriting its attributes and behaviors, representing a enemy character in the game. The class has a static attribute called all_enemies to keep track of all instances of the Enemies class created.
  - The attribute of most importance in this class is behavior_tree, which helps control what the instance will do when calling its update method. On top of that, other attributes such as maximum_movement_per_frame, movement_counter, can_reach_player, target_player, can_reach_item, target_item, target_box, and is_just_placed_bomb help the behavior tree function well.
  - The Enemy class overrides update method and its update rate is controlled with a view to enhancing the quality of the gameplay. It also contains some getter and setter methods. On top of that, the Enemy class has handy methods such as find_nearest_box(), find_nearest_grid(), get_distance_to_target, move_towards_target(), and move_to_grid() being called in the behavior tree.
- Class Button
  - The Button class represents a simple button element that can be drawn on a Pygame surface.
- Class Text
  - The Text class epresents a simple text element that can be drawn on a Pygame surface.
- Class GameMode
  - The class GameMode controls the whole game flow. An instance is created when the main.py is running, and three methods are called in sequence.
  - start_menu(self) displays the game's start menu. It includes buttons for starting the game and draws the title. Waiting for user interaction to start the game.
  - game_mode() initializes game elements, such as players, enemies, bombs, and the game map. After that, it enters the game loop, where it continuously checks for events, updates the game state, and redraws the screen. Also it handles player input for movement and bomb placement and checks for game-ending conditions (player or enemy death) and breaks out of the loop.
  - end_menu() displays the game over menu, showing a message based on the game outcome (win or lose) and it includes buttons for restarting the game or quitting.
- MapGenerator
  - The class MapGenerator handles the map creation. An instance is created in GameMode.gamemode() and its method create_map() is called.
  - create_map() method first reads the .txt file in the directory Map. Then it creates instances of the derived classes of the Obstacles class. The type of instance is decided by the character given by the .txt file.

C. *Other Features*

- Behavior Tree
  1) Tree Structure
  2) Types of the Condition Nodes

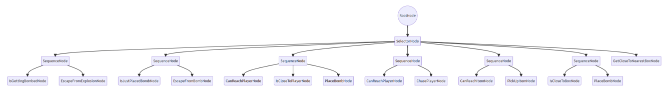

Fig. 4. Behavior Tree of the Enemy Class

a) IsGettingBombedNode: Checks if the enemy is currently being bombed.
b) IsJustPlacedBombNode: Checks if the enemy has just placed a bomb.
c) IsCloseToPlayerNode: Checks if the enemy is close to a player.
d) CanReachPlayerNode: Checks if the enemy can reach a player.
e) CanReachItemNode: Checks if the enemy can reach an item.
f) IsCloseToBoxNode: Checks if the enemy is close to a box.

3) Types of the Action Nodes
   a) EscapeFromExplosionNode: Moves the enemy away from the nearest grid that is not exploded.
   b) EscapeFromBombNode: Moves the enemy away from a bomb.
   c) PlaceBombNode: Places a bomb if the enemy is allowed to.
   d) PickUpItemNode: Moves towards and picks up an item.
   e) ChasePlayerNode: Moves towards and chases a player.
   f) GetCloseToNearestBoxNode: Moves towards the nearest box.
4) Execution: The Behavior Tree is executed in a loop during the enemy's update cycle. Each node's execute method is called, and the tree traverses according to the defined logic. Actions are performed based on the conditions met during traversal.
5) Purpose: The Behavior Tree is designed to create a flexible and self-controlled system for Enemy class in this game. By combining different types of nodes and defining specific conditions and actions, it allows for complex and dynamic behaviors in response to changing game conditions. The tree's hierarchical structure and node types enable the Enemy instances to make decisions based on various criteria and perform diverse actions in a game environment.

- A* Algorithm (a_star(start, goal) function)
  1) Parameter: start: The starting node and goal: The goal node.
  2) Return: The path (list of coordinates) from the starting node to the goal node.
  3) Implementation: First, initializes open and closed lists, creates start and goal nodes. Second, uses a priority queue (heap) for the open list. Third, Explores neighboring nodes, calculates costs, and updates the open list. Then, continues until the goal is reached or open list is empty. Last, If a path is found, reconstructs and returns the path; otherwise, returns None.
  4) Purpose: The A* algorithm efficiently explores the grid space, and it is used in the game for the enemy to find its target and path, considering both the cost to reach a node from the start (g cost) and a heuristic estimate of the remaining distance to the goal (h cost). The Node class represents grid nodes, the heuristic function calculates distances, and the a_star function finds the optimal path using the A* algorithm. The nearest_grid_not_exploded function helps avoid exploding cells when finding nearby grids.

III. Appendix

A. *Outlook of the Game*

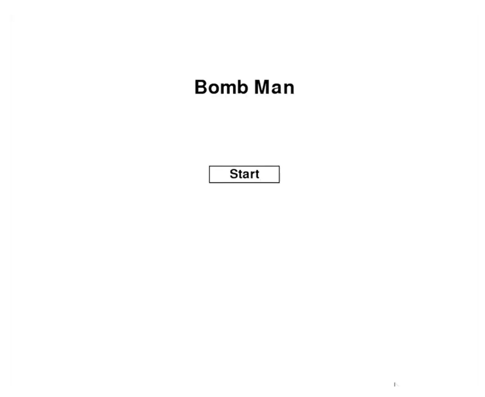

Fig. 5. Starting Menu

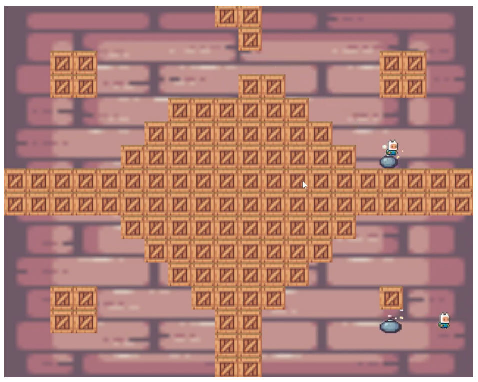

Fig. 6. Game Play

IV. Reference

References

[1] ChatGPT
[2] https://github.com/veldahung/group21_project/tree/main
[3] https://www.pygame.org/wiki/tutorials
[4] https://www.pygame.org/wiki/tutorials

Fig. 7. Ending Menu