Team 7

Gameplay Documentation

HIGH LEVEL OVERVIEW:

The game is called by the gameplay manager which creates an instance of GameBoad. GameBoard calls on GameConfiguration to create the randomized board and takes the output information and uses it to populate the game model with the map(consisting of blocks, exit and walls) as well as the sprites(enemies and bomberman). The GameBoard then creates an instance of InputHandler which takes care of the player input and starts the gameloop in which it calls an update method on all the game elements currently on the board and then calls render methods on all the gameobjects in order to draw them all on the board.

GameObjects consist of two types, immoveable (GameSolid) and moveable (GameActor). GameObject is an abstract class which is implemented by GameSolid and GameActor. GameSolid is used for creating the blocks and walls while daughter classes to GameSolid take care of the special cases of the door, bombs and powerups. GameActors splits into Enemies and Bomberman. Enemies contains a component, AI and a clone method which allows each enemy to act a spawner for itself. Enemy has further daughter classes for each of the enemy types available, each populated with their own Image and game stats which are unique to the enemy type.

Bomberman is a representation of bomberman, the class creates bombs by creating instances of the bomb class and is able to communicate to remotely detonated the bombs through user actions.Powerups are managed by making modifidations to the bomberman state variables (lives, speed, wall walking, number of bombs, explosion radius). Bomb destruction is managed by an observer pattern. Bombs are implemented as observable and every gameobject subscribes to the bomb class and is alerted when a bomb is detonated, the GameObject class is then responsible to determine if a game element has been destroyed by a bomb.

Movement in the game is designed around a command design patern, the InputHandler listens to the keyboard constantly and when a key is pressed the buttons call the associated command which then commands the GameActors to perform the action. This setup allows all GameActors to be controlled via these controls and allows for the same control framework to be used for both the AI and the user. In the case of the enemies, the AI outputs command objects which drive the enemy units. This allows for easy swapping of AIs, any AI can drive drive any gameactor.

Game loading and saving are managed through the gamestate. On load, gamestate is passed in from the save/load system and repopulates the map, the enemies and bomberman. Saving the game entails returning the gamestate object back up the chain to the save/load system.

CLASS: BombermanManager

Class is the entry point to the game engine and deals with handling flow of information before the game board takes over, creates an instance of gameboard and populates the bomberman JFrame.

CLASS: GameBoard

This class manages the GameBoard and is responsible for drawing the map and maintaining all the elements on the map (list activeElements). The main game loop can be found in this class. Has a gamestate type which was specified externally to this diagram, the gamestate is passed between login,highscore, save and gameplay system, it contains userName, score, lives, bomberman and the initial map information.

METHODS

gameBoard(GameState): Constructor which takes in a saved game state and reloads the game by setting the gameState to the loaded gameState.

proccessInput(): Function call which checks for input from the user via a key listener and input from the AI.

update(): Called by the game loop to run all the update logic on the gameobjects, deals purely with model information. Calls an update method on every element of the gameobject list which updates all the elements to the next game frame.

render(): Draws the next gameframe to screen by calling the draw methods on all the game elements and draws the HUD (score/time/lives).

shortestPathBetweenObjects(GameObject,GameObject): returns the shortest path between two objects on the game board.

createNewGameObject(int,int,string): creates a new object at coordinates x,y of enemy type string.

killGameObject(): removes the game object from the list of activeGameObjects on the board.

CLASS:GameConfiguration

Contains methods and information relevant to each level. Contains enemy types, number of enemies and number of blocks required for each level and is responsible for generating a

CLASS: GameState

This class stores all the information on the game including score, remining lives, bomberman's state and the initial gameboard(assuming we randomly generate each level, elsewise this element is not required)

ABSTRCT CLASS: GameObject

Abstract class to the game elements, includes the coordinates and visibility status. Contains the x and y coordinates of

METHODS

draw(): virtual draw method which all classes need to implement, allows for the gameboard to call the draw() method on all game elements, thus rendering each instance responsible for it's own drawing.

hasColided(): Checks for colision between the calling object and another object on the board.

CLASS: GameActor

Contains information relevant to game objects which move. Boolean value wallPass keeps track if the game element should be able to walk through destroyable walls.

METHODS

moveLeft/Right/Up/Down(): Implement movements commands on game actors

CLASS: Bomberman

Class takes care of all the information relevant to bomberman specifically.

METHODS:

placeBomb(): method which adds a bomb to the gameworld.

detonateBomb(): command which calls for the bomb to detonate

Class:Enemy

Stores information relevant to each enemy, contains a special component EnemyAI which stores the profile for different enemy AIs.

METHODS:

clone(): clones the current enemy, acts as a unit spawner for when a bomb is detonated on top of the exit.

CLASS: GameSolid

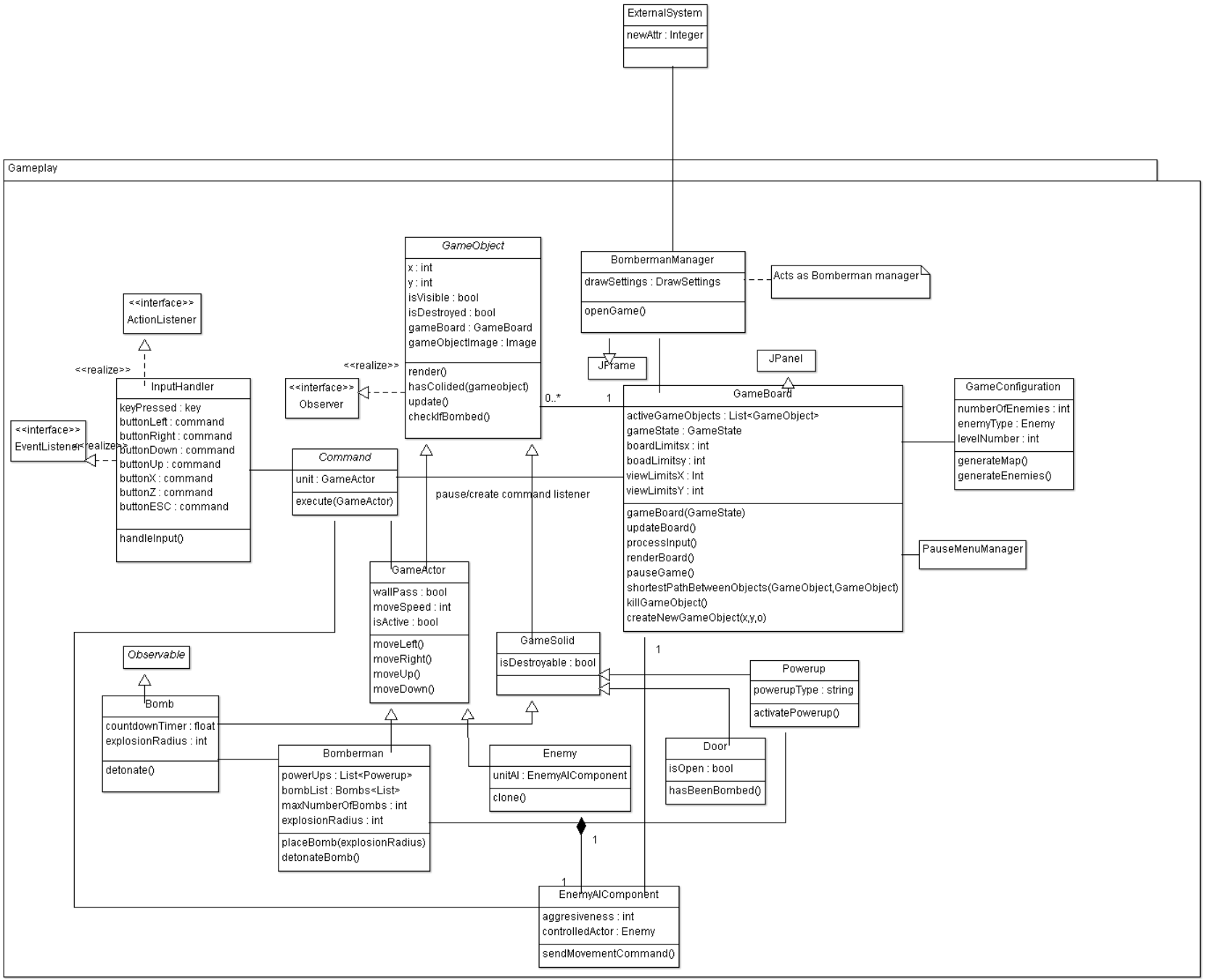
Implements the abstract class GameObject and is used to create bricks and walls.

CLASS: Bomb/Powerup/Door

Special cases of GameSolid which implement the bomb, powerup and door game objects. Powerups call a function when activate which is a simple switch statement. The door is either open or close and if the door has been blasted with a bomb.

CLASS: Command(ABSTRACT) /InputHandler/EnemyAI

Three classes implement the command pattern. The inputHandler watches for keystrokes and communicates to the Command class to execute commands on an moveable game entity, this allows both InputHandler and EnemyAI to use the same setup to move the enemy and bomberman


ExternalSystem
newAttr : Integer
Gameplay
GameObject
x : int
y : int
isVisible : bool
isDestroyed : bool
gameBoard : GameBoard
gameObjectImage : Image
render()
hasColided(gameobject)
update()
checkIfBombed()
BombermanManager
drawSettings : DrawSettings
openGame()
Acts as Bomberman manager
JFrame
JPanel
<<interface>>
ActionListener
<<realize>>
InputHandler
keyPressed : key
buttonLeft : command
buttonRight : command
buttonDown : command
buttonUp : command
buttonX : command
buttonZ : command
buttonESC : command
handleInput()
<<realize>>
<<interface>>
Observer
0..*
1
GameBoard
activeGameObjects : List<GameObject>
gameState : GameState
boardLimitsx : int
boadLimitsy : int
viewLimitsX : Int
viewLimitsY : int
gameBoard(GameState)
updateBoard()
processInput()
renderBoard()
pauseGame()
shortestPathBetweenObjects(GameObject,GameObject)
killGameObject()
createNewGameObject(x,y,o)
GameConfiguration
numberOfEnemies : int
enemyType : Enemy
levelNumber : int
generateMap()
generateEnemies()
<<interface>>
EventListener
<<realize>>
Command
unit : GameActor
execute(GameActor)
pause/create command listener
PauseMenuManager
GameActor
wallPass : bool
moveSpeed : int
isActive : bool
moveLeft()
moveRight()
moveUp()
moveDown()
GameSolid
isDestroyable : bool
1
Observable
Powerup
powerupType : string
activatePowerup()
Bomb
countdownTimer : float
explosionRadius : int
detonate()
Bomberman
powerUps : List<Powerup>
bombList : Bombs<List>
maxNumberOfBombs : int
explosionRadius : int
placeBomb(explosionRadius)
detonateBomb()
Enemy
unitAI : EnemyAIComponent
clone()
Door
isOpen : bool
hasBeenBombed()
1
1
1
EnemyAIComponent
aggresiveness : int
controlledActor : Enemy
sendMovementCommand()