Bilkent University

Department of Computer Engineering

CS 319- Object-Oriented Software Engineering

System Design Report

War of Domination

Group 1-H

Akant Atılgan

Kazım Ayberk Tecimer

Unas Sikandar Butt

Supervisor: Eray Tüzün

Contents

1.Introduction 4

1.1 Purpose of the System 4

1.2 Design Goals 4

1.2.1 Criteria 4

End User Criteria 4

Usability: 4

Efficiency: 4

Reliability: 5

Modifiability: 5

2. System Architecture 6

2.1 Subsystem Decomposition 6

2.2 Hardware/ Software Mapping 9

2.3 Persistent Data Management 9

2.4 Access Control and Security 9

2.5 Boundary Conditions 10

3. Subsystem Services 11

3.1. User Interface Subsystem UserInterface Subsystem 11

Class definitions: 12

3.1.1 MainFrame Class 12

3.1.2 MenuView Class 14

3.1.3 SettingsMenu Class 15

3.1.4 PauseMenu Class 17

3.1.5 MapView Class 18

3.1.6 GameView Class 19

3.2 Game Management Subsystem 21

3.2.1 Game Manager 22

3.2.2 Map Manager 25

3.2.3 Input Manager 27

3.2.4 Inventory Manager 28

3.2.5 Action Manager 29

3.2.6 Attack Manager 30

3.2.7 Movement Manager 31

3.2.8 Sound Manager Class 32

3.2.9 File Manager 33
3.3 Game Object Subsystem 34
3.3.1 Map Class 35
3.3.2 Game Object 36
3.3.3 Non-Stationary 37
3.3.4 Weapons 37
3.3.5 Grenade 38
3.3.6Bullet 39
3.3.7 Knife 39
3.3.8 Character 40
3.3.9 Stationary 41
3.3.10 Passable 42
3.3.12 Obstacles 43
3.3.13 Bush, Road, Water 44
3.3.14 Non-Passable 44
4. Low-level Design 45
4.1 Object Design Trade-Offs 45
Understandability vs Functionality: 45
Memory vs Maintainability: 45
Development Time vs User Experience: 45
4.2 Packages 46
4.2.1 org.newdawn.slick.command 46
4.2.2 org.newdawn.slick.gui 46
4.2.3 org.newdawn.slick.tiled 46
4.2.4 org.newdawn.slick.state 46
4.2.5 org.newdawn.slick.loading 46
4.3 Class Interfaces 46
4.3.1 InputListener 46
4.3.2 KeyListener 47
4.3.3 MouseListener 47
4.3.4 ControllerListener 47
5. References & Glossary 47

1.Introduction

1.1 Purpose of the System

War of Domination is a 2D turn-based strategy game which aims to provide users with an enjoyable gaming experience. The game consists of a user-friendly user interface with How-to-Play tutorials and basic settings options. Users can choose to play in a single player mode or compete with each other using the multiplayer mode. The main aim of the player in the game is to control his characters and lead them to defeat the opposing team. The game will come with multiple maps which the user can choose from to play from a selection of mods. The tools given to the player are a rifle, grenade and a knife. Other weapons or power-ups may be found on the map.

1.2 Design Goals

1.2.1 Criteria

End User Criteria

Usability:

The game will provide a friendly user interface which can be navigated with ease by the use of both the mouse and the keyboard. The game will also provide a tutorial so that the players can learn the game play, rules and objectives. Furthermore, the controls of the game can be modified by the players to suit their own preferences.

Efficiency:

Efficiency will be one of the systems key goals. The system will run moderate graphics so that the game performance is kept smooth. To ensure this all the Image files will be compressed to the limit as to not hold any extra data. This will also ensure efficiency in the management of data.Also the maps will be pre-loaded while opening the game; so initialization of maps will take minimal time.

Reliability:

Reliability is also one of the main concerns of the system. Game applications have to be very reliable as there is always a lot of processing going on in all the subsystems. Any sort of glitch or crash will lose the players their current achievements and score. To make the system reliable the game will be extensively tested at all the stages of development,

Modifiability:

The system is being developed using the principles of object oriented programming. This practice will ensure modifiability. System will be divided into meaningful subsystems which will all handle different tasks of the system. The MVC architectural style that will be used will ensure that any modification to a subsystem will not cause other systems to malfunction.

2. System Architecture

2.1 Subsystem Decomposition

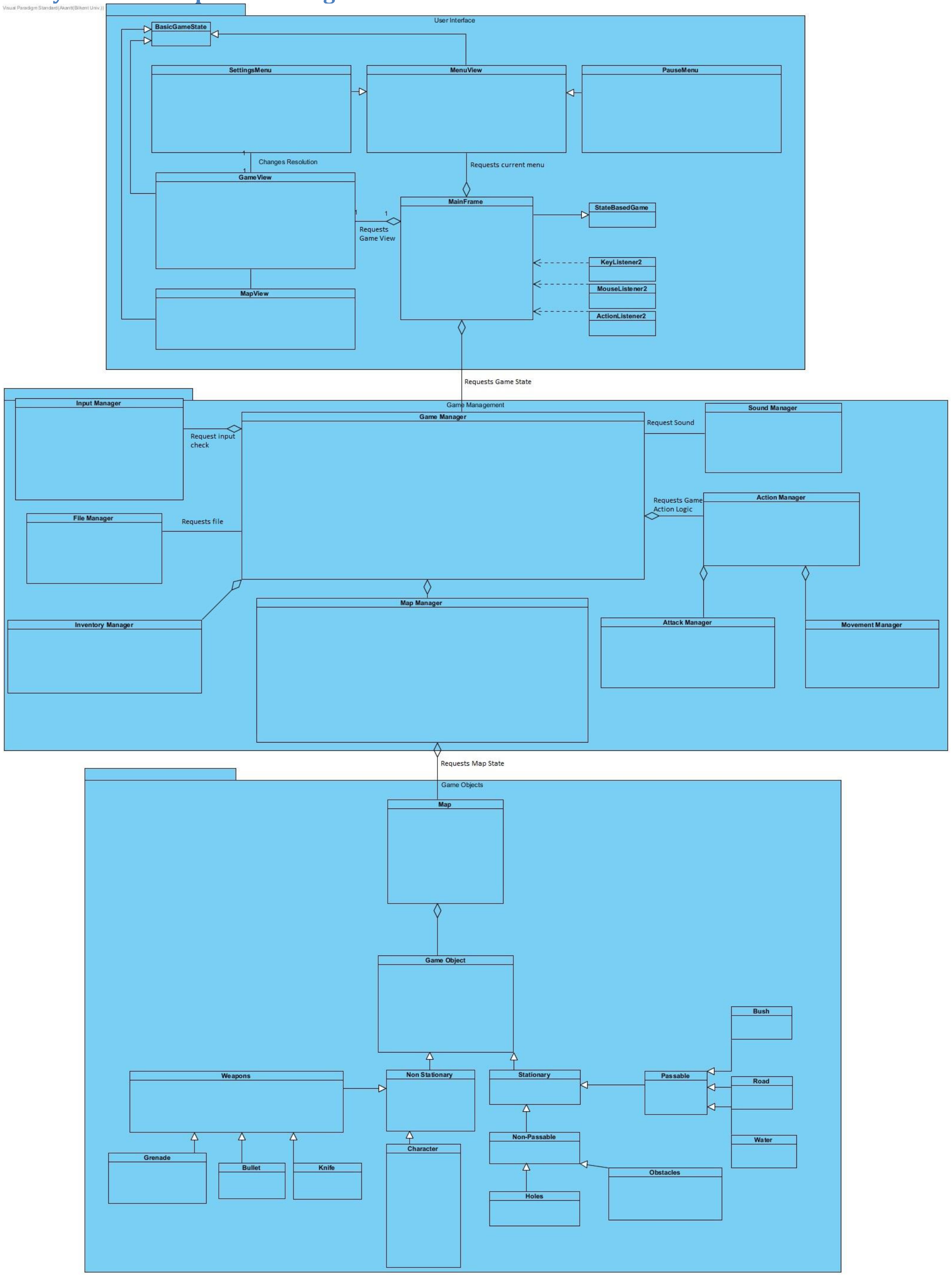
In this part, our system is divided into pieces called subsystems. Purpose of the division is to reduce the coupling between different subsystems of the main system and to increase the efficiency of relations between components. As dividing the system into parts, when the system need to be changed it will be much easy to modify and extend the system.

During the decomposition of the system, we use the Model-View-Controller architectural pattern as it fits into our program. There will be 3 subsystems that compose the our system. First one is the model classes, basically object classes stand for the representing the Model (e.g. game objects, map). Secondly, our classes related with UI will compose the View pattern as they provide interface for the users. Lastly, management classes will compose the Controller as they update and maintain the game. The detailed explain of these classes will be given below.

In the decomposition stage, we try to divide our system to subsystems by looking at classes's functionalities. To do so, we divide our system into three subsytems as we think that it is the efficient way of structuring the system. Each system has its own responsibilities, and it does not conflict with the responsibilities of other systems. More, sub-systems are in relation to each other. But this relationship is not direct, communication flows as in the design of the façade are based on one class. We decided to use façade design pattern as it decreases the complexity of the class relations and allows us to solve any errors related with the system. To elabore this, as it can be observed from the next figures, Our View subsystem is in touch with the Game management system by using the game manager class.

As a result, system decomposition will provide us smoothness and efficient system with less connection. By utilizing from the façade pattern, we will be able to extend our system with facing less errors.

Subsystem Decomposition Diagram



2.2 Hardware/ Software Mapping

War of Domination will be implemented in Java programming language; more specifically we will use Slick2d libraries (check glossary). Hence, as software requirement, JRE(Java Runtime Environment), which can be found across many devices since Java is a widespread environment, Slick2d library, and LWJGL library are required to system to be executed by other users.

More, War of Domination will require a computer which includes keyboard, mouse, monitor, hardware equipments. Keyboard and mouse will be used for the input relation between user and the system. Users will be able to control the menu, characters, map with both keyboard and mouse. By having the requirements mentioned above, users can able to run the game.

2.3 Persistent Data Management

Our system does not require any complex data storage system or any databases. It will store the game instances, characters, maps, game modes in the memory of the computer. We are storing our persistent data in files and folders and users will be able to modify them. Background images and music could be easily changed by adding them to the game folder. Hard-disk drive is used to store all these information.

2.4 Access Control and Security

As the game we are creating is single-player and must be downloaded. After loading the game and starting the system user can play the game. In our system, we do not have any implementation of an authentication system. More, for the security of the user, as the system has nothing to do with the internet and database we will not be able to access any of the users' data. For the security of system we divided our system to many smaller components in order to make them trace and debug easily, which increase the stability of the system. Also utilizing from the encapsulation, we make our system as black-box, that is other people will not be able to modify our system as they wanted.

2.5 Boundary Conditions

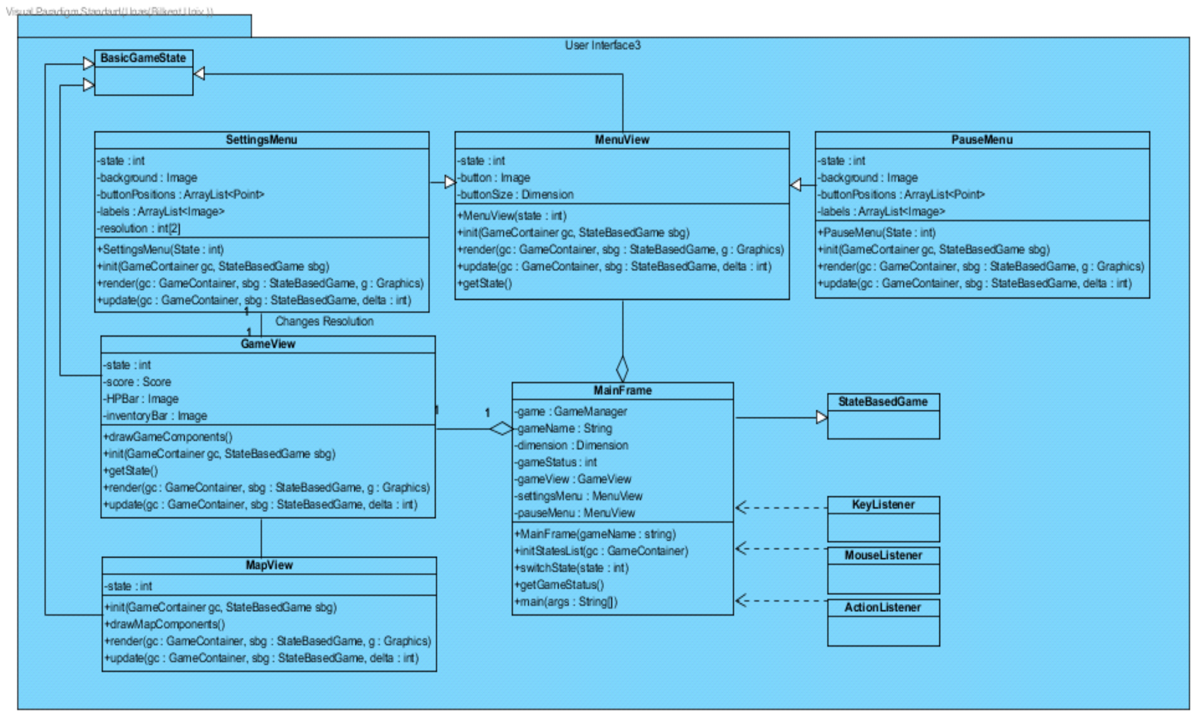
Our system will not have any .exe extention but game will have an executable jar and the will be executed from .jar.

Our system can be terminated by clicking “Exit” button in the main menu. Another way of exiting the system in the game is first pause the game and click exit button. In case of the death of one players, game will return to main menu screen.

If an unexpected event occurs during the game stage, system will safely exit without harming anything on the system.

3. Subsystem Services

3.1. User Interface Subsystem UserInterface Subsystem

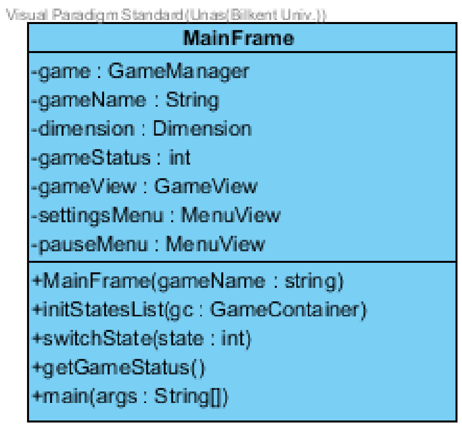


User Interface3
BasicGameState
SettingsMenu
-state : int
-background : Image
-buttonPositions : ArrayList<Point>
-labels : ArrayList<Image>
-resolution : int[2]
+SettingsMenu(State : int)
+init(GameContainer gc, StateBasedGame sbg)
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)
MenuView
-state : int
-button : Image
-buttonSize : Dimension
+MenuView(state : int)
+init(GameContainer gc, StateBasedGame sbg)
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)
+getState()
PauseMenu
-state : int
-background : Image
-buttonPositions : ArrayList<Point>
-labels : ArrayList<Image>
+PauseMenu(State : int)
+init(GameContainer gc, StateBasedGame sbg)
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)
1
Changes Resolution
1
GameView
-state : int
-score : Score
-HPBar : Image
-inventoryBar : Image
+drawGameComponents()
+init(GameContainer gc, StateBasedGame sbg)
+getState()
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)
1
1
MainFrame
-game : GameManager
-gameName : String
-dimension : Dimension
-gameStatus : int
-gameView : GameView
-settingsMenu : MenuView
-pauseMenu : MenuView
+MainFrame(gameName : string)
+initStatesList(gc : GameContainer)
+switchState(state : int)
+getGameStatus()
+main(args : String[])
StateBasedGame
KeyListener
MouseListener
ActionListener
MapView
-state : int
+init(GameContainer gc, StateBasedGame sbg)
+drawMapComponents()
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)

The UI subsystem will be in charge of handling the view part of the game's MVC architectural style. This subsystem will deal primarily with boundary objects, and relay the user's actions to the game controller.

Class definitions:

3.1.1 MainFrame Class



Inheritance: Inherits StateBasedGame class. The StateBasedGame class is provided in the slick library.

Interfaces: This class will implement the KeyListener, MouseListener and ActionListener interfaces.

Class Function: This class will be in charge of creating the primary window for the game. This window will hold different states at each point of the game. This class also holds the main function which will be responsible to start the game. This class is also the only link between the View and the Controller packages.

Attributes:

- -game : GameManager GameManager is a class from the Controller package. The MainFrame will communicate with the controller through this object and update accordingly.

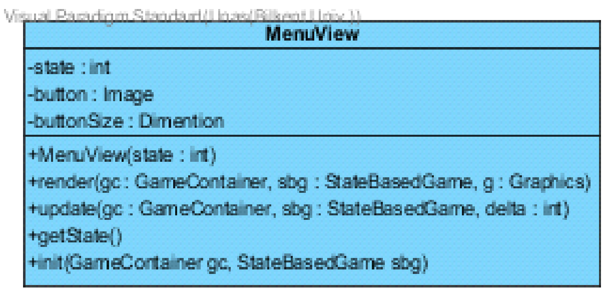
- -gameName : String This will be the title of the game which will be displayed in the title bar of the window.
- -dimension : Dimension Dimension class will be imported from Java's awt library. This object will hold the dimensions of the game window namely, width and height.
- -gameStatus : int This integer value will determine which state the game is currently in. For example main menu, in game, settings or pause menu.
- -gameView : GameView This object will hold the game state. The MainFrame will display this state when the player is playing the game.
- -settingsMenu : MenuView Similar to the gameView this object will hold the state of the game where the user is in the settings menu.
- -pauseMenu : MenuView This state will be displayed when the player will pauses the game while playing.

Methods:

- initStatesList (gc : GameContainer) This method will initialize all the states of the game by calling the init() method of all of them. Also this method will insert the initial game state into the main window. This method is provided in the StateBasedGame class which is inherited and it is overridden here.
- switchState (state : int) The switch state method will be called by the MainFrame class to switch the state. For example switching from gameView state to the pauseMenu state.

- getGameStatus () This method will return the state that the game is currently in. This will be called to make decisions that are based on the state that the game is currently in.

3.1.2 MenuView Class


MenuView
-state : int
-button : Image
-buttonSize : Dimention
+MenuView(state : int)
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)
+getState()
+init(GameContainer gc, StateBasedGame sbg)

Inheritance: This class will inherit the BasicGameState class provided in the slick library

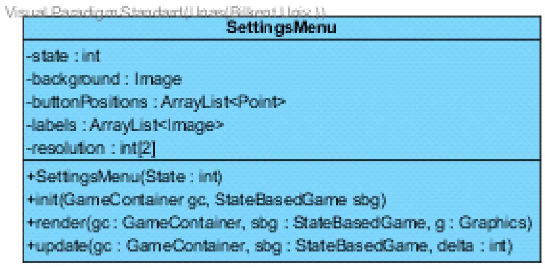
Attributes:

- -state : int This will basically used to store the state ID of the view. Which will be used to distinguish between different states of the game.
- -button : Image This image will be a plain button which will be used multiple times on the screen to signify different options in the menu. This button will then be overwritten on to display a string e.g. “Settings”.
- buttonSize: Dimension This will store the standard button size for all the buttons in the game.

Methods:

- init (GameContainer : gc, StateBasedGame sbg) This method is provided in the parent class and is overridden here. This method will initialize the view of this state. For example the button positions and background image will be initialized here.
- render (gc : GameContainer, sbg : StateBasedGame, g : graphics) This method will be in charge of drawing all of the screen elements.
- update (gc : GameContainer, sbg : StateBasedGame, delta : int) This method will update key values in the game in every time unit delta. After all the values have been updated render() will be called.
- getState() This method will return the state ID of the view

3.1.3 SettingsMenu Class


Visual Paradigm Standard(Ilyas(Bilkent Univ.))
SettingsMenu
-state : int
-background : Image
-buttonPositions : ArrayList<Point>
-labels : ArrayList<Image>
-resolution : int[2]
+SettingsMenu(State : int)
+init(GameContainer gc, StateBasedGame sbg)
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)

Inheritance: This class will inherit the MenuView State

Attributes:

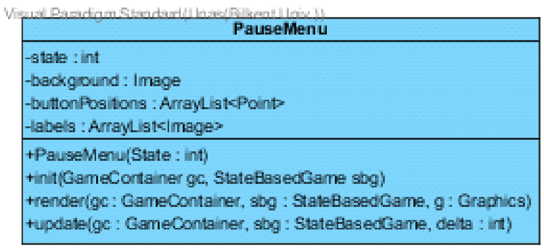
- -state : int This will basically used to store the state ID of the view. Which will be used to distinguish between different states of the game.
- -background : Image Background will store the image file that will be displayed first when the state is active

- -buttonPositions : ArrayList<Point> This list will hold all locations of the buttons so that we can use it to draw the button provided in the parent class.
- -labels : ArrayList<Image> This will store the labels of all the buttons as Images. These will be drawn after the buttons are drawn.
- -resolutions : int[2] This will store the current resolution of the game. E.g. 640x480

Methods:

- init (GameContainer : gc, StateBasedGame sbg) This method is provided in the parent class and is overridden here. This method will initialize the view of this state. For example the button positions and background image will be initialized here. Overridden from the parent class.
- render (gc : GameContainer, sbg : StateBasedGame, g : graphics) This method will be in charge of drawing all of the screen elements. Overridden from the parent class.
- update (gc : GameContainer, sbg : StateBasedGame, delta : int) This method will update key values in the game in every time unit delta. After all the values have been updated render() will be called. Overridden from the parent class.

3.1.4 PauseMenu Class


PauseMenu
-state : int
-background : Image
-buttonPositions : ArrayList<Point>
-labels : ArrayList<Image>
+PauseMenu(State : int)
+init(GameContainer gc, StateBasedGame sbg)
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)

Inheritance: This class will inherit the MenuView State

Attributes:

- -state : int This will basically used to store the state ID of the view. Which will be used to distinguish between different states of the game.
- -background : Image Background will store the image file that will be displayed first when the state is active
- -buttonPositions : ArrayList<Point> This list will hold all locations of the buttons so that we can use it to draw the button provided in the parent class.
- -labels : ArrayList<Image> This will store the labels of all the buttons as Images. These will be drawn after the buttons are drawn.

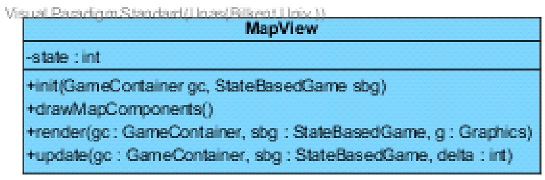
Methods:

- init (GameContainer : gc, StateBasedGame sbg) This method is provided in the parent class and is overridden here. This method will initialize the view of this state. For example the button positions and background image will be initialized here. Overridden from the parent class.

- render (gc : GameContainer, sbg : StateBasedGame, g : graphics) This method will be in charge of drawing all of the screen elements. Overridden from the parent class.

- update (gc : GameContainer, sbg : StateBasedGame, delta : int) This method will update key values in the game in every time unit delta. After all the values have been updated render() will be called. Overridden from the parent class.

3.1.5 MapView Class



Visual Paradigm Standard(Ilhan(Bilkent Univ.))
MapView
-state : int
+init(GameContainer gc, StateBasedGame sbg)
+drawMapComponents()
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)

Inheritance: This class will inherit the BasicGameState class.

Class Function: This class's view will be drawn before the game view. The game view will then be drawn over this view.

Attributes:

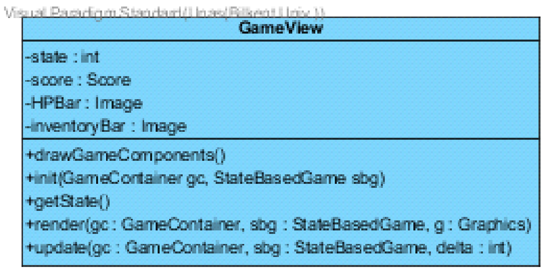
- -state : int This will basically used to store the state ID of the view. Which will be used to distinguish between different states of the game.

Methods:

- init (GameContainer : gc, StateBasedGame sbg) This method is provided in the parent class and is overridden here. This method will initialize the view of this state. For example the map tiles and obstacles will be initialized here. Overridden from the parent class.

- drawMapComponents() This method will be called by the render method to draw the map components that keep changing for the animation. For example the grass tiles and breaking obstacles.
- render (gc : GameContainer, sbg : StateBasedGame, g : graphics) This method will be in charge of drawing all of the screen elements. Overridden from the parent class.
- update (gc : GameContainer, sbg : StateBasedGame, delta : int) This method will update key values in the game in every time unit delta. After all the values have been updated render() will be called. Overridden from the parent class.

3.1.6 GameView Class


GameView
-state : int
-score : Score
-HPBar : Image
-inventoryBar : Image
+drawGameComponents()
+init(GameContainer gc, StateBasedGame sbg)
+getState()
+render(gc : GameContainer, sbg : StateBasedGame, g : Graphics)
+update(gc : GameContainer, sbg : StateBasedGame, delta : int)

Inheritance: This class will inherit the BasicGameState class.

Attributes:

- -state : int This will basically used to store the state ID of the view. Which will be used to distinguish between different states of the game.
- -score : Score This object will store the users score points
- -HPBar : Image This object will store the image for the player's current HP
- -inventoryBar : Image This object will hold the image for the player's current inventory

Methods:

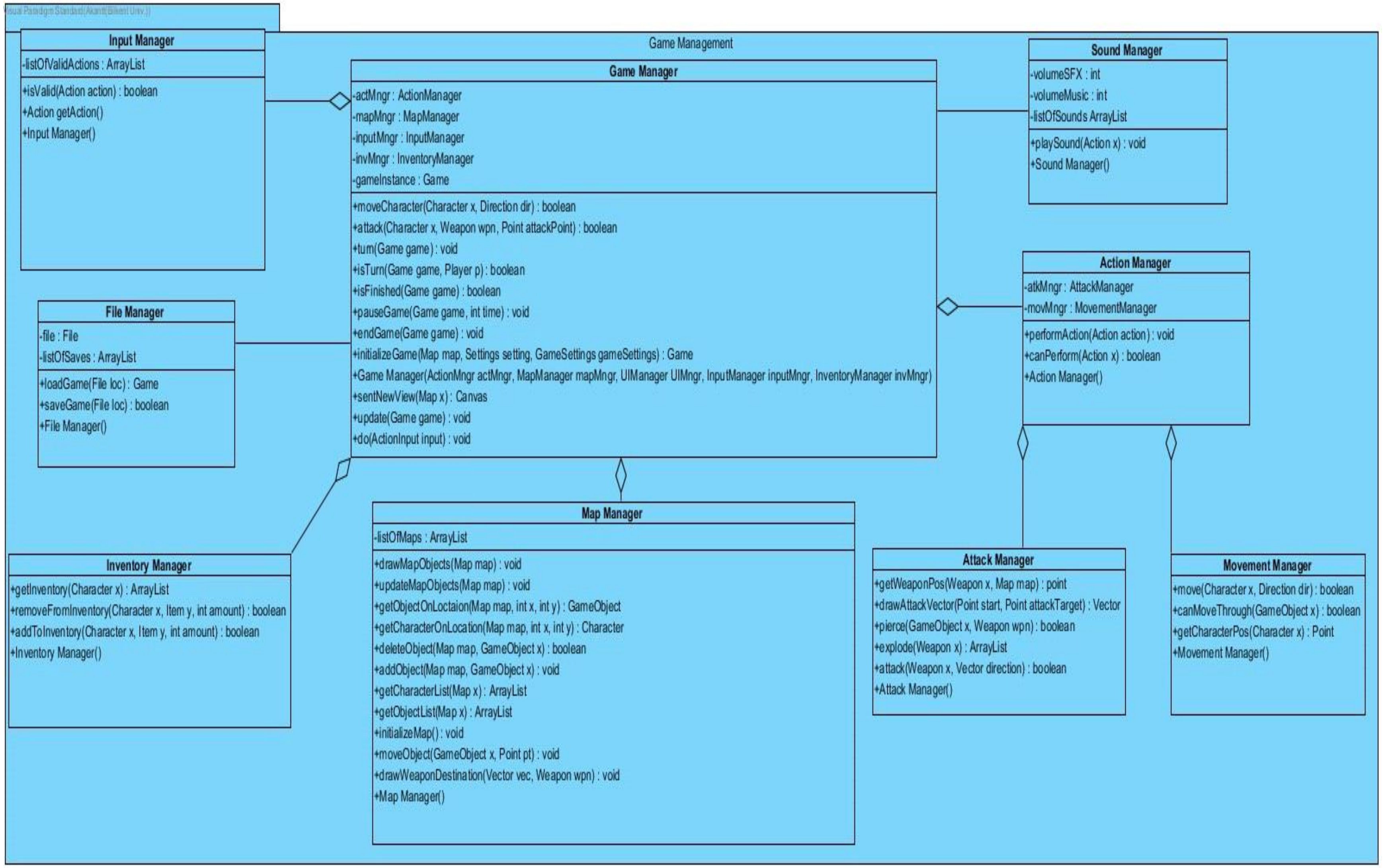
- init (GameContainer : gc, StateBasedGame sbg) This method is provided in the parent class and is overridden here. This method will initialize the view of this state. For example the initial player positions and weapons will be initialized here. Overridden from the parent class.

- drawGameComponents() This method will be called by the render method to draw the game components that keep changing for the animation. For example the player moving and shooting.

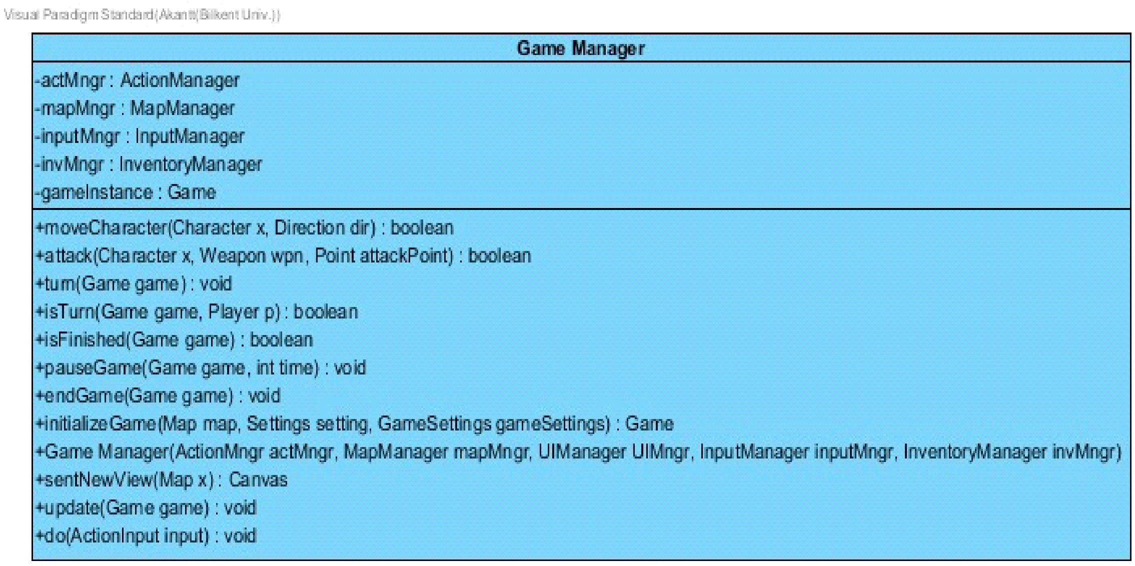
- render (gc : GameContainer, sbg : StateBasedGame, g : graphics) This method will be in charge of drawing all of the screen elements. Overridden from the parent class.

- update (gc : GameContainer, sbg : StateBasedGame, delta : int) This method will update key values in the game in every time unit delta. After all the values have been updated render() will be called. Overridden from the parent class

3.2 Game Management Subsystem


Game Management
Input Manager
-listOfValidActions : ArrayList
+isValid(Action action) : boolean
+Action getAction()
+Input Manager()
Game Manager
-actMngr : ActionManager
-mapMngr : MapManager
-inputMngr : InputManager
-invMngr : InventoryManager
-gameInstance : Game
+moveCharacter(Character x, Direction dir) : boolean
+attack(Character x, Weapon wpn, Point attackPoint) : boolean
+turn(Game game) : void
+isTurn(Game game, Player p) : boolean
+isFinished(Game game) : boolean
+pauseGame(Game game, int time) : void
+endGame(Game game) : void
+initializeGame(Map map, Settings setting, GameSettings gameSettings) : Game
+Game Manager(ActionMngr actMngr, MapManager mapMngr, UIManager UIMngr, InputManager inputMngr, InventoryManager invMngr)
+sentNewView(Map x) : Canvas
+update(Game game) : void
+do(ActionInput input) : void
Sound Manager
-volumeSFX : int
-volumeMusic : int
-listOfSounds ArrayList
+playSound(Action x) : void
+Sound Manager()
Action Manager
-atkMngr : AttackManager
-movMngr : MovementManager
+performAction(Action action) : void
+canPerform(Action x) : boolean
+Action Manager()
File Manager
-file : File
-listOfSaves : ArrayList
+loadGame(File loc) : Game
+saveGame(File loc) : boolean
+File Manager()
Inventory Manager
+getInventory(Character x) : ArrayList
+removeFromInventory(Character x, Item y, int amount) : boolean
+addToInventory(Character x, Item y, int amount) : boolean
+Inventory Manager()
Map Manager
-listOfMaps : ArrayList
+drawMapObjects(Map map) : void
+updateMapObjects(Map map) : void
+getObjectOnLoctaion(Map map, int x, int y) : GameObject
+getCharacterOnLocation(Map map, int x, int y) : Character
+deleteObject(Map map, GameObject x) : boolean
+addObject(Map map, GameObject x) : void
+getCharacterList(Map x) : ArrayList
+getObjectList(Map x) : ArrayList
+initializeMap() : void
+moveObject(GameObject x, Point pt) : void
+drawWeaponDestination(Vector vec, Weapon wpn) : void
+Map Manager()
Attack Manager
+getWeaponPos(Weapon x, Map map) : point
+drawAttackVector(Point start, Point attackTarget) : Vector
+pierce(GameObject x, Weapon wpn) : boolean
+explode(Weapon x) : ArrayList
+attack(Weapon x, Vector direction) : boolean
+Attack Manager()
Movement Manager
+move(Character x, Direction dir) : boolean
+canMoveThrough(GameObject x) : boolean
+getCharacterPos(Character x) : Point
+Movement Manager()

3.2.1 Game Manager



```
Game Manager
-actMngr : ActionManager
-mapMngr : MapManager
-inputMngr : InputManager
-invMngr : InventoryManager
-gameInstance : Game
+moveCharacter(Character x, Direction dir) : boolean
+attack(Character x, Weapon wpn, Point attackPoint) : boolean
+turn(Game game) : void
+isTurn(Game game, Player p) : boolean
+isFinished(Game game) : boolean
+pauseGame(Game game, int time) : void
+endGame(Game game) : void
+initializeGame(Map map, Settings setting, GameSettings gameSettings) : Game
+Game Manager(ActionMngr actMngr, MapManager mapMngr, UIManager UIMngr, InputManager inputMngr, InventoryManager invMngr)
+sentNewView(Map x) : Canvas
+update(Game game) : void
+do(ActionInput input) : void
```

Attributes:

- -ActionManager actMngr: This attribute holds a ActionManager to perform some required actions to the game state; for example moving, shooting. Calculations of these action results are done in action manager.

- -MapManager mapMngr: This attribute holds a MapManager to perform some of positional changes to game objects as game progresses.

- -Input Manager inputMngr: This attribute holds a input manager to make sure specified hotkey/ mouse actions that requires a change in game state are catched properly and sent to game manager.

- -InventoryManager invMngr: An instance of inventorymanager is created inside gamemanager to manipulate character inventories easly by accessing to inventory managers helper methods.

- Game gameInstance: This is the main game instance; the main game states are held here.Ex: turn state; map state;win state etc.

Methods:

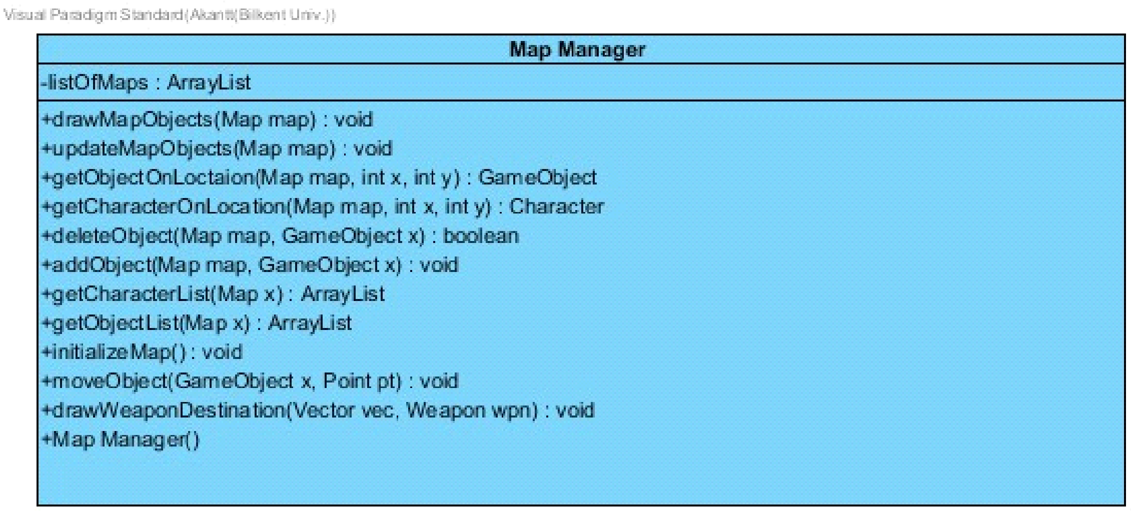
- +boolean moveCharacter(Character x,Direction dir): Determines if the specified character is eligible to move at specified direction or not.(Turn system should not be overritten.)It moves the character if it's possible.
- +boolean attack(Character x,Weapon wpn,Point attackPoint): Determines if the specified character is eligible to attack at specified point with specified weapon. If the character can perform an attack; then it creates an attacking animation on map by getting the attack vector with the help of action manager by passing the point as a parameter.
- +void turn(Game game): Passes the turn to the other player in the queue on the determined game. When the turn is passed map view also gets slight changes to make sure user(s) are aware of the turn passing.
- +boolean isTurn(Game game,Player p): Determines if the specified player has the turn at that moment at the specified game.
- +boolean isFinished(Game game):Determines if the specified victory / defeat condition has been reached or not. This is called each time a game switches turns to make sure no other turns are played when the game is already over. If it returns true game manager switches view to game finish screen with the help of “endGame” method.

- +void pauseGame(Game game,Time t): Pauses the game in a halt state. Every kind of action derived animation on the map gets freezed immidiately until pause period is over.
- + void endGame(Game game): Ends the game and calculates / prints necessary scores for the game finish screen.
- +Game initializeGame(Map map,Settings setting,GameSettings gameSetting):
- This method starts the game logic state by getting map, dimension, user settings(hotkeys, sound etc.) and gamesettings(victory condition game mod player count etc.). This method returns a game object as a result ready to be played. It also calls initialize methods of other managers to make sure other properties of game are initialized such as UI and map rendering.
- +Canvas sendNewView(Map x): This method will send the updated view of the map. This is called each time logic of the game is updated or user creates an input(ex: move mouse , click, attack etc.)
- +void update(Game game): Updates the state of the map by calling other managers methods.
- +void do(ActionInput input): Gets user input and sends it to input manager to handle it; if it is a input for a valid action; game manager will change the game state with the help of action manager.

Constructors:

- +Game Manager(ActionMngr actMngr, MapManager mapMngr, UIManager UIMngr, InputManager inputMngr, InventoryMngr invMngr): Default Constructor for this class. When game manager is created it also creates it's helper managers to distribute game logic weight.

3.2.2 Map Manager



```
Map Manager
-listOfMaps : ArrayList
+drawMapObjects(Map map) : void
+updateMapObjects(Map map) : void
+getObjectOnLoctaion(Map map, int x, int y) : GameObject
+getCharacterOnLocation(Map map, int x, int y) : Character
+deleteObject(Map map, GameObject x) : boolean
+addObject(Map map, GameObject x) : void
+getCharacterList(Map x) : ArrayList
+getObjectList(Map x) : ArrayList
+initializeMap() : void
+moveObject(GameObject x, Point pt) : void
+drawWeaponDestination(Vector vec, Weapon wpn) : void
+Map Manager()
```

Attributes

- ArrayList<Map> listOfMaps: This attribute holds a list of maps available for user to pick from.

Methods

- +drawMapObjects(Map map): This method will render map objects; it's very similar to render() function.

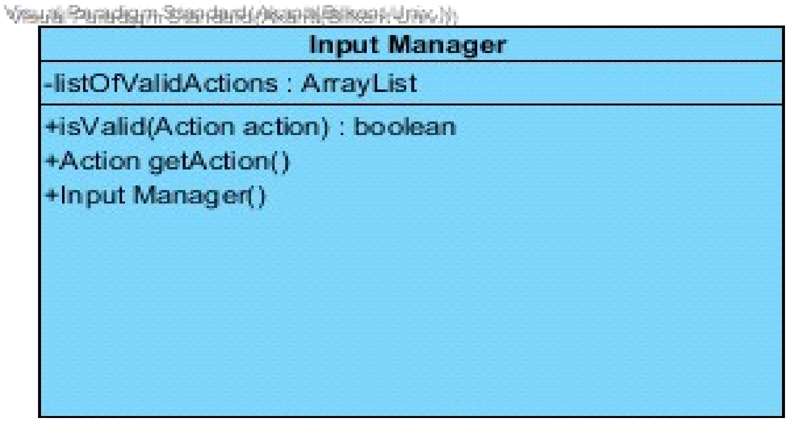
- +updateMapObjects(Map map): This method will update map objects properties like life,position etc. After this method it always calls drawMapObjects method as well to make sure updates are displayed.

- +GameObject getObjectOnLocation(Map map,int x,int y): This method will return the game object on the specified location as an object.
- +Character getCharacterLocation(Map map ,int x,int y): This method will return the character object on the specified location as an object.
- +boolean deleteObject(Map map,GameObject x): This method will delete the certain game object from the map. It will return true if it can find & delete the object.
- +addObject(Map map,GameObject x): This method will add a certain object to a map.(No need to specift location etc. because game object is already created with a location.
- +ArrayList<Character> getCharacterList(Map x): This method will return a list of characters
- +ArrayList<Game Object> getObjectList(Map x):This method will return a list of all objects on the specified map.
- +initializeMap(): This method will initialize all maps by putting in the objects in the specified locations.
- +moveObject(Game Object x Point pt):This method allows the change of a objects position without any restriction. So other classes should check the logic behind a movement before calling this.
- +drawWeaponDestination(Vector vec,Weapon wpn):This method will take an array of points to move the object one by one on these points to simulate an “projectile motion” on the screen.

-Constructors

- +Map Manager():Default constructor for this class; it will create a few maps when this constructor is called.

3.2.3 Input Manager


Visual Paradigm Standard(Akarsu(Bilkent Univ.))
Input Manager
-listOfValidActions : ArrayList
+isValid(Action action) : boolean
+Action getAction()
+Input Manager()

-Attributes

- -ArrayList<Action> listOfValidActions: This attribute will hold a list of valid actions specified from the user beforehand.

-Methods

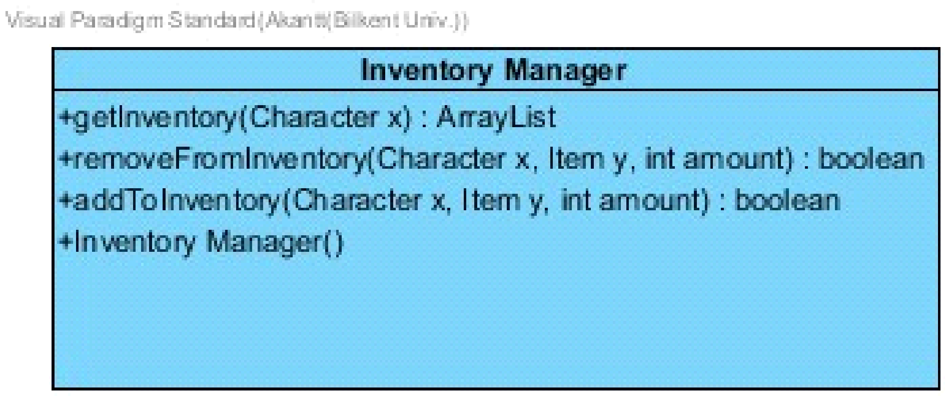
- +boolean isValid(Action action): This method will return true if the user requested a valid action with his/her inputs.

- +Action getAction():This method will create an action based on the user input. This allows game manager to update game state based on the actions.

-Constructors

- +Input Manager(): Default constructor for this class. Creates a list of valid actions(developers list these not user, user will define hotkeys not actions defined by these hotkeys!) when created.

3.2.4 Inventory Manager


Visual Paradigm Standard(Akanti(Bilkent Univ.))
Inventory Manager
+getInventory(Character x) : ArrayList
+removeFromInventory(Character x, Item y, int amount) : boolean
+addToInventory(Character x, Item y, int amount) : boolean
+Inventory Manager()

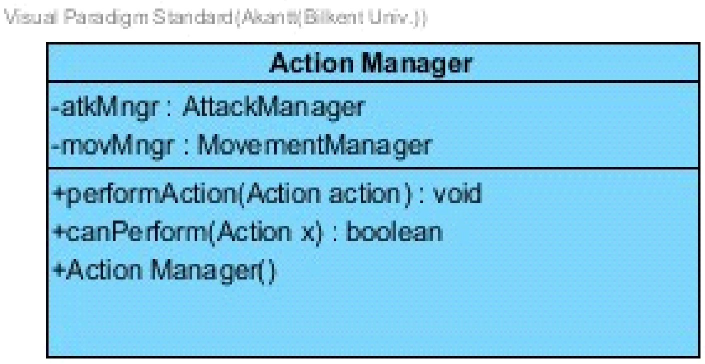
-Methods

- +ArrayList<Inventory> getInventory(Character x): This method will return the inventory of a character.
- +boolean addToInventory(Character x,Item y,int amount): This method lets the game manager to add a item to a specified character.
- +boolean removeFromInventory(Character x,Item y,int amount):This method will be called from game manager if a character consumes an item from inventory. It will delete a specified item for a specified amount.

-Constructors

- +Inventory Manager(): Default constructor for the class. Allows the creation and calls of methods.

3.2.5 Action Manager


Visual Paradigm Standard(Akant(Bilkent Univ.))
Action Manager
-atkMngr : AttackManager
-movMngr : MovementManager
+performAction(Action action) : void
+canPerform(Action x) : boolean
+Action Manager()

-Attributes

- -AttackManager atkMngr: This attribute holds an instance of attackmanager to help with action performing and checks.
- -MovementManager movMngr: This attirubute holds an instance of movementmanager to help with

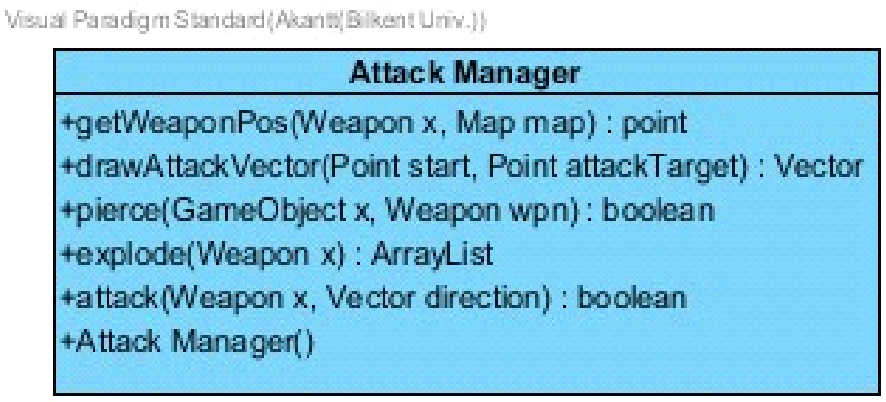
-Methods

- +void performAction(Action action):This method will take an action from game manager that is created by input manager. It will process the action with the help of attack and movement managers and return the values that game manager needs to process the action.
- +boolean canPerform(Action x):This method will return if the action requested is possible to perfom at that state.

-Constructors

- Action Manager():Default constructor for action manager; this instance will be created in game manager to access this classes methods.

3.2.6 Attack Manager


Visual Paradigm Standard(Akant(Bilkent Univ.))
Attack Manager
+getWeaponPos(Weapon x, Map map) : point
+drawAttackVector(Point start, Point attackTarget) : Vector
+pierce(GameObject x, Weapon wpn) : boolean
+explode(Weapon x) : ArrayList
+attack(Weapon x, Vector direction) : boolean
+Attack Manager()

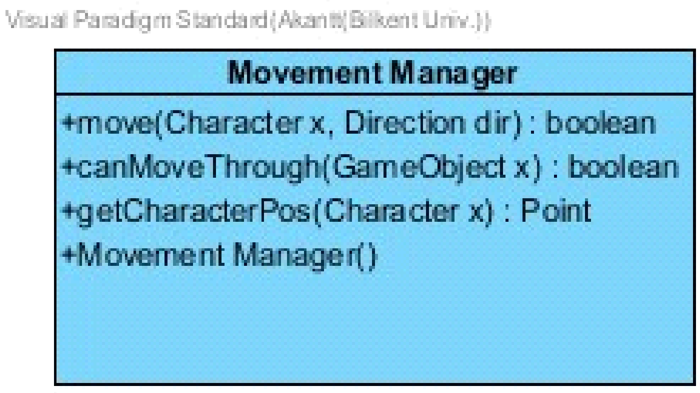
-Methods

- +point getWeaponPos(Weapon x,Map map): Gets the position of the weapon at the specified map. This is a helper method to create the attack vector.
- +Vector drawAttackVector(Point start,Point attackTarget): This method creates an array of points on the specified path between 2 points; that will be used later on the map manager to create an animation of “projectile motion”.
- +boolean pierce(Game Object x,Weapon wpn): Determines if the object will pierce the object; if it does it will determine how much of the damage potential of the weapon is lost while piercing.
- +ArrayList<Image> explode(Weapon x): This array will create an array list of images that will be the .gif of an explosion on the screen of the specified weapon. This will be sent back to game manager to be displayed on map.
- +boolean attack(Weapon x,Vector direction):This will be the attacking action which will return true if the action is performed successfully.If it is not handled successfully it will return false so it will be easier to find bugs in the logic.

-Constructors

- +Attack Manager(): Default constructor for the attack manager class that will allow other classes to use the private methods.

3.2.7 Movement Manager


Visual Paradigm Standard(Akant(Bilkent Univ.))
Movement Manager
+move(Character x, Direction dir) : boolean
+canMoveThrough(GameObject x) : boolean
+getCharacterPos(Character x) : Point
+Movement Manager()

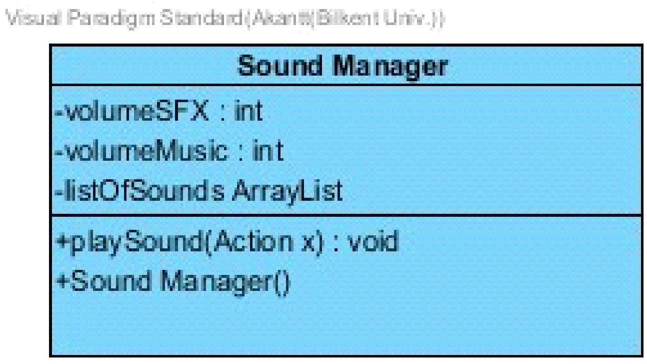
-Methods

- +boolean move(Character x,Direction dir): This will return true if the character can move at that direction and the game manager will do the action since action manager completed the task successfully with the help of this class.
- +boolean canMoveThrough(Game Object x): This will allow characters to move through passable objects by comparing their “isPassable” boolean and returning the value; so they “canMoveThrough” the object.
- +Point getCharacterPos(Character x):This is a helper function that will help with collusion calculation. Also it will be used to determine if a character can move through something

-Constructors

- Movement Manager(): Default Constructor for this class. It will help action manager to access this class's private methods.

3.2.8 Sound Manager Class



-Attributes

- -int volumeSFX:This attribute is set during the settings screen and it will allow us to play pre-defined sounds at a certain level between 0 and 100.
- -int volumeMusic:This attribute works same way as volumeSFX; only difference is this will be used for music sound.
- -ArrayList<Sound> listOfSounds:This attribute will contain all lists of pre-defined sounds like weapon explosions movement etc.

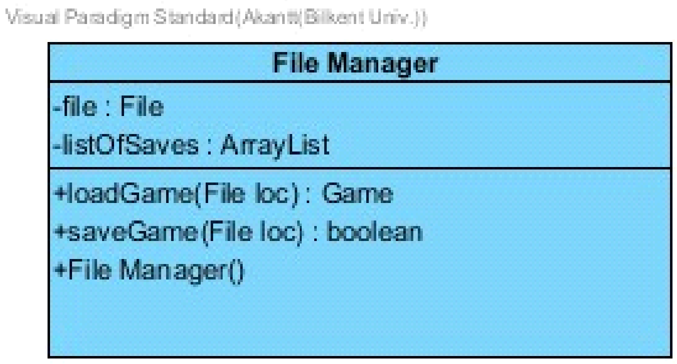
-Methods

- +void playSound(Action x):This method will play a specific sound based on the action performed.

-Constructors

- +Sound Manager():This is the default constructor for this class. It will initialize the list of pre-defined sounds when created.

3.2.9 File Manager


Visual Paradigm Standard(Akant(Bilkent Univ.))
File Manager
-file : File
-listOfSaves : ArrayList
+loadGame(File loc) : Game
+saveGame(File loc) : boolean
+File Manager()

-Attributes

- -File file: This will hold the main folder to save/load games.
- ArrayListOfSavedGames<File> listOfSaves:This attribute holds the locations of all saved games for the user to pick from if they want to load a game.

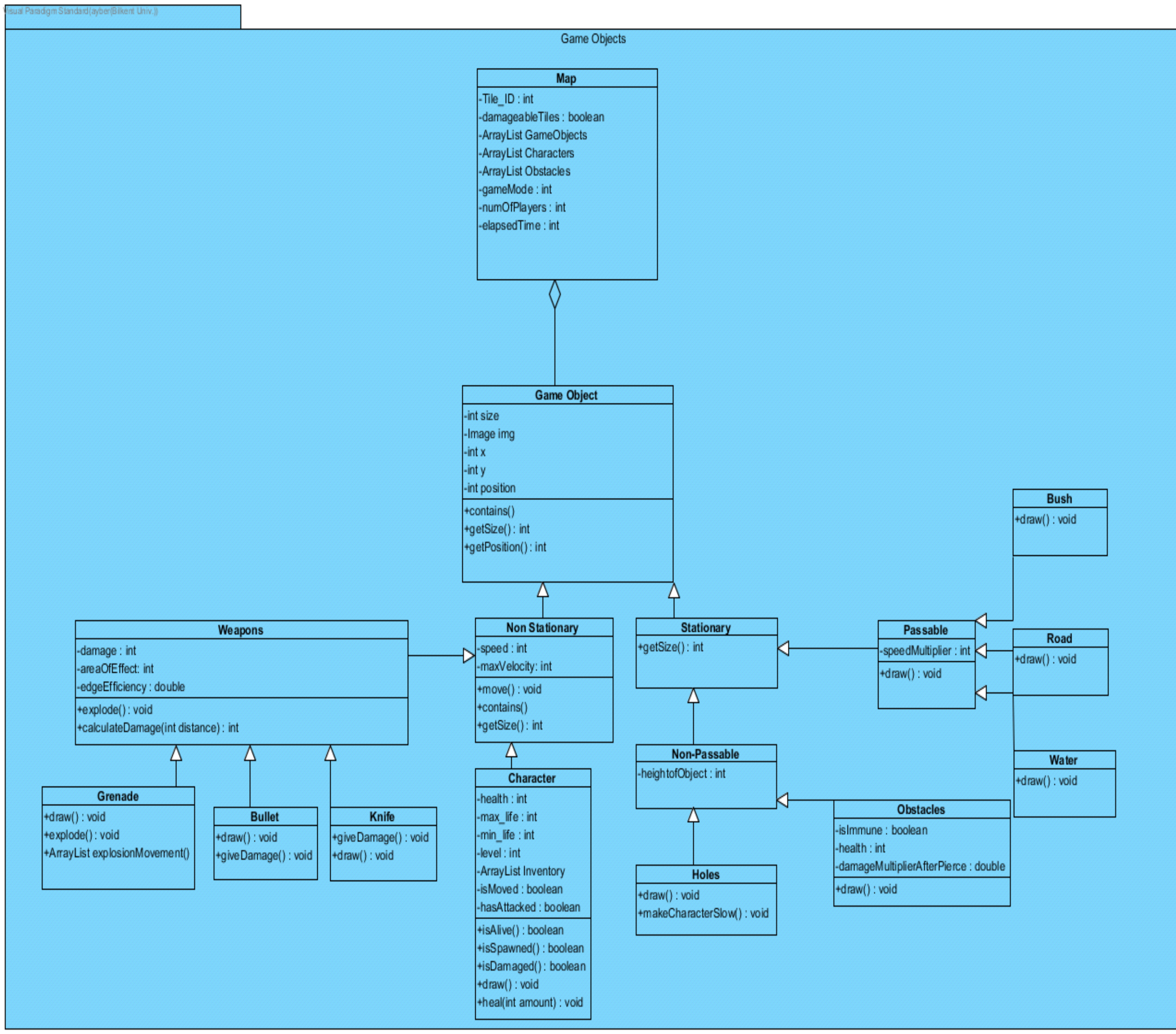
-Methods

- +Game loadGame(File x):This method will load the game at the specified location. It will send all the serialized info to the game manager to load it.
- +Game saveGame(File x):This method will serialise the list of objects and the game state; save it on a folder to be loaded later on when user wants.

-Constructors

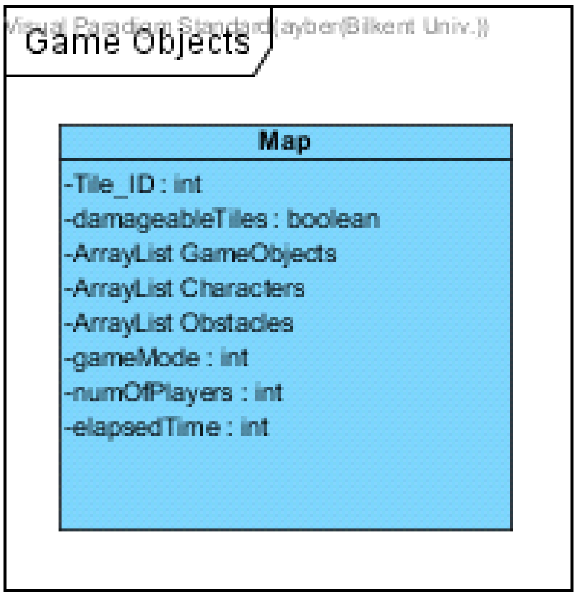
- File Manager():Default constructor for this class. Allows the game manager to access it's properties.

3.3 Game Object Subsystem



Game Objets Subsystem consists of 16 classes and is responsible for the holding the identities of the objects and communicate with the GameManagement class which is controller class. The purpose of this subystem is to abstractions of the models in the system. modifications on the models will not be done in the model classes. The main objects of the game are Non-Stationary (able to move), and Stationary (static) objects. Non-Stationary objects are characters, weapons (grenade, bullet, knife), Stationary objects are non-passable(holes, obstacles), passable(bush, road, water). According to the objects user is going to able to see the information about game objects.

3.3.1 Map Class



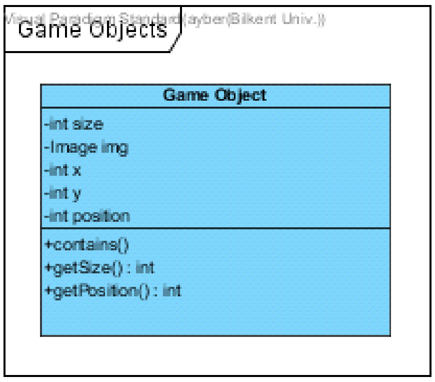
Attributes:

- int Tile_ID: This attribute stands for the categorize the different tiles (bush, road, water)
- boolean damageableTiles: This attribute returns the type of tile either damageable or non-damageable
- Arraylist GameObjects: This Arraylist holds the game objects
- Arraylist Characters: This character arraylist holds the characters
- Arraylist Obstacles: This Obstacles arraylist holds the obstacles
- int gameModes: Since there will be more than one game mode, this integer determines the which game mode will be shown on the map.

- int numOfPlayers: This attribute returns the number of players in the game.

- int Elapsed Time: This attribute determines the time that is passed.

3.3.2 Game Object



Attributes:

- int size : This attribute returns the size of the game object

- int position: This attribute returns the position of the game object

- int x: This attribute holds the x coordinate of the game object

- int y: This attribute holds the y coordinate of the game object

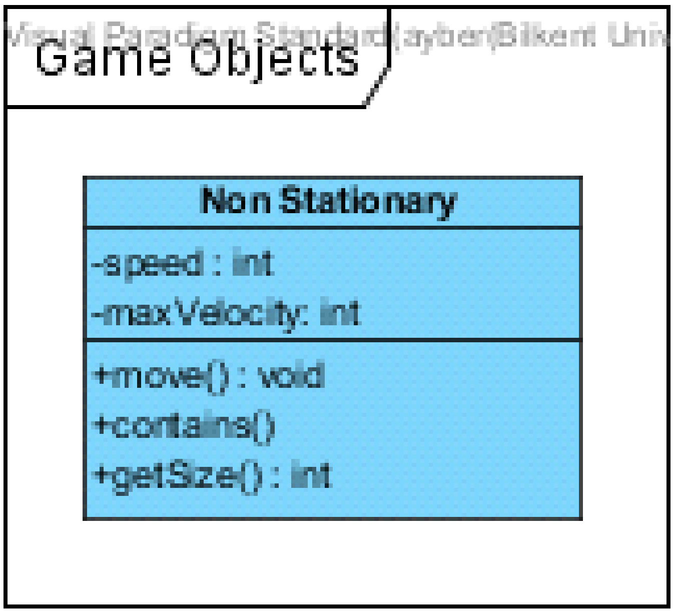
- image img: This attribute is the image of the object

- Operations

- int getSize(): this operation returns the size of the game Object

- int getPosition(): this operation returns the position of the game object

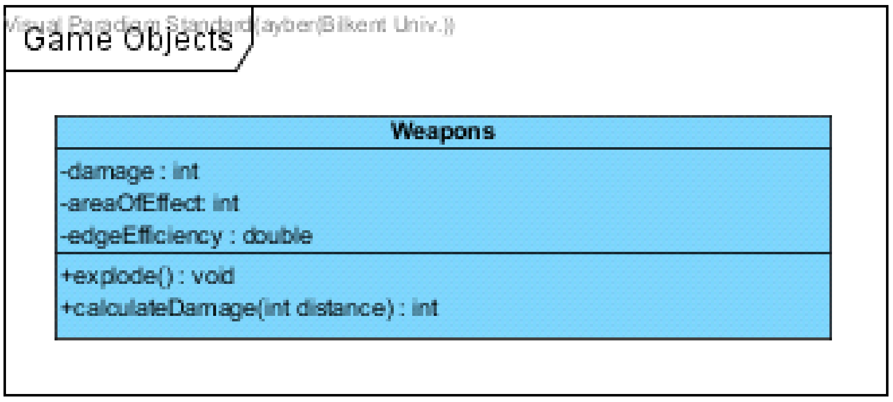
3.3.3 Non-Stationary



Attributes

- int speed: this attribute returns the speed of the non-stationary
- int maxVelocity: this attribute returns the maximum velocity of the non-stationary
- Operations:
- void move(): moves the non-stationary
- int getSize(): returns the size of the non-stationary

3.3.4 Weapons



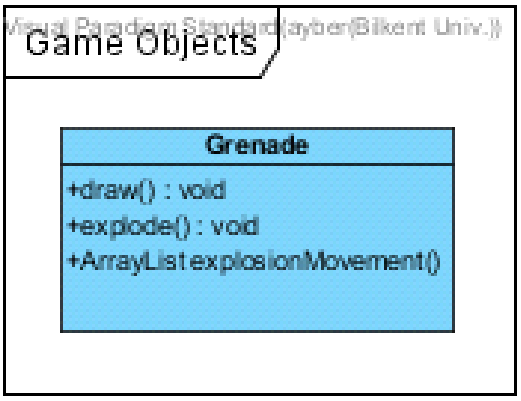
Attributes

- int damage : returns the damage that the weapon gives
- int areaOfEffect: returns the area that the weapon effects
- int edgeEfficiency: returns the value of efficiency as weapon get close to edges

Methods:

- void explode(): weapon explodes
- int calculateDamage(int distance): this operation returns the damage which is calculated, distance affects the damage

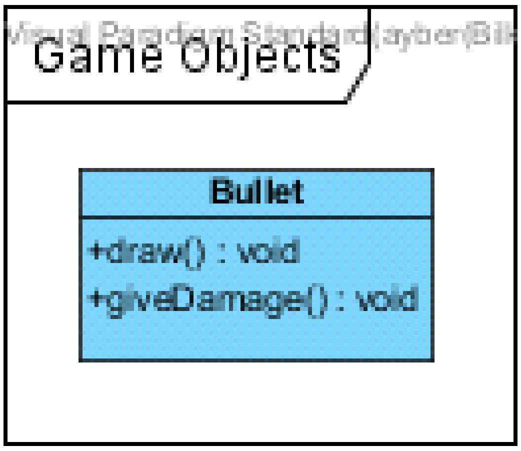
3.3.5 Grenade


Visual Paradigm Standard(ayber(Bilkent Univ.))
Game Objects
Grenade
+draw() : void
+explode() : void
+ArrayList explosionMovement()

Methods:

- void draw(): draws the grenade
- void explode(): grenade explodes
- ArrayList ExplosionMovement(): this operation return the arraylist image to show the explosion of the grenade

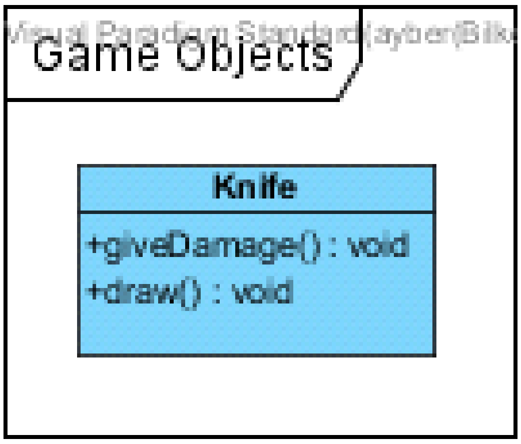
3.3.6Bullet



Methods:

- void draw(): draws the bullet
- void giveDamage(): bullet gives damage to other characters

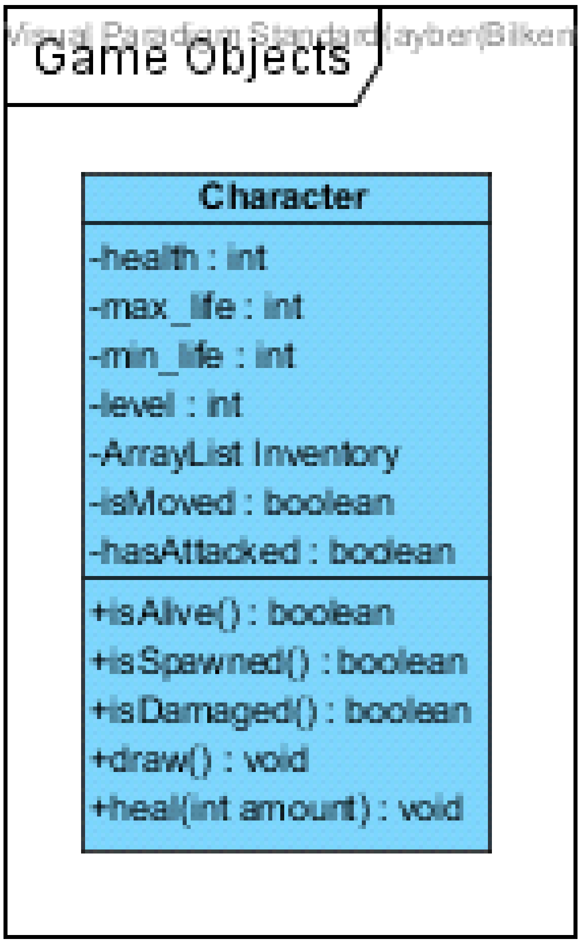
3.3.7 Knife



Methods:

- void giveDamage(): knife gives damage to other characters
- void draw(): draws the knife

3.3.8 Character



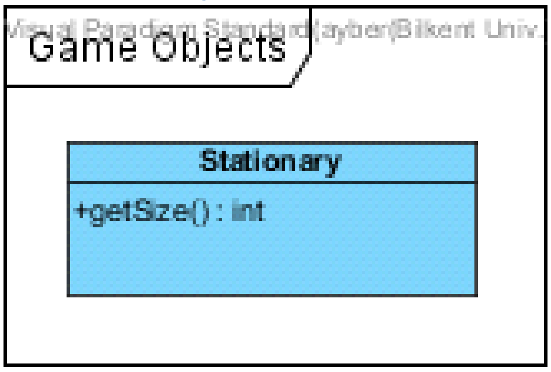
Attributes

- int health: this attribute returns the amount of life of the character
- int max_life: this attribute returns the maximum number of life of the character
- int min_life: this attribute returns the minimum number of life of the character
- int level: returns the level of the character
- ArrayList Inventory: holds the inventory of the character in the ArrayList
- boolean isMoved: returns the true if character is moved otherwise returns false

Methods:

- boolean isAlive(): this method checks whether character is alive or not
- boolean isSpawned(): this method checks whether character is spawned or not
- boolean isDamaged(): this method checks whether character is damaged or not
- void draw(): draws the character
- void heal(int amount): this method allows character to increase the health

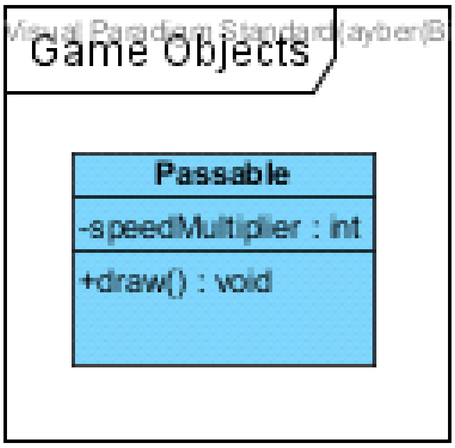
3.3.9 Stationary



Methods:

- int getSize(): returns the size of the stationary

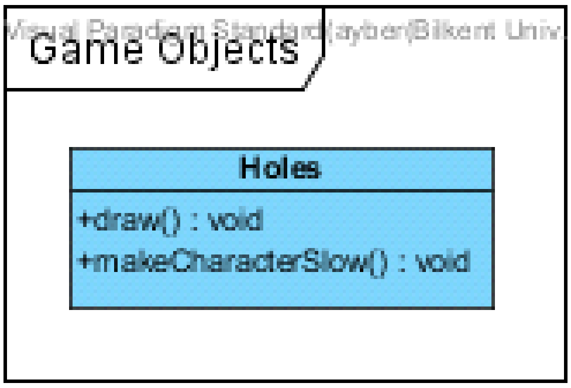
3.3.10 Passable



Attribute

- int speedMultiplier: this returns the speed multiplier among the passables as it differs among them.

3.3.11 Holes

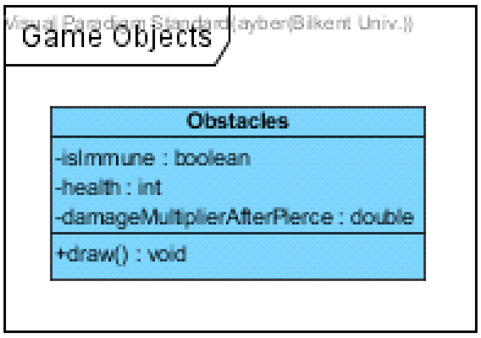

Visual Paradigm Standard(ayberi(Bilkent Univ.
Game Objects
Holes
+draw() : void
+makeCharacterSlow() : void

Methods:

- void draw(): draws the hole

- void makeCharacterSlow(): holes make character to slow. It decreases the speed of the character.

3.3.12 Obstacles


Visual Paradigm Standard(ayber(Bilkent Univ.))
Game Objects
Obstacles
-isImmune : boolean
-health : int
-damageMultiplierAfterPierce : double
+draw() : void

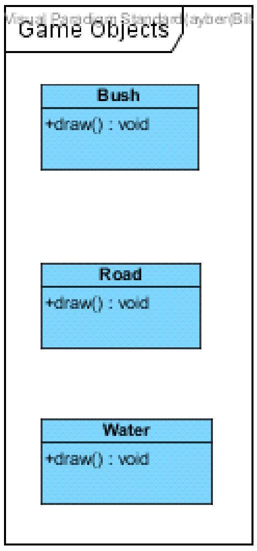
Attributes

- boolean isImmune: returns the immunability of the obstacle (damageable or not)
- int health: returns the health of the object
- double damageMultiplierAfterPiece: it returns the value of damage multiplier after it pierced

Methods:

- void draw(): draws the obstacle

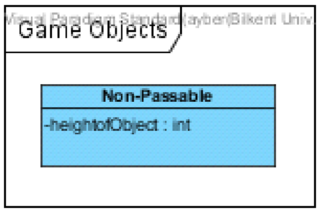
3.3.13 Bush, Road, Water



Methods:

- void draw(): draws the bush, road, and water

3.3.14 Non-Passable



Attribute

heightofObject: int returns the objects' height

4. Low-level Design

4.1 Object Design Trade-Offs

Understandability vs Functionality:

The purpose of our system is to allow users easily learn the system and adopt it. Hence, we try to prevent the complexity of the game, we tried to eliminate the confusing and complicated features of the game, so that, the players will be easily play the game.

Memory vs Maintainability:

During the system design phase, we try to remove the common parts of the system and make use of models (abstractions). The purpose of the design is to maintain the game objects. As modeling allows us to abstract the common attributes as much as possible, it would be very straightforward to maintain the the system that will lead user to have better utilization. By utilizing from these features (common attributes) of subsystem and subclasses, when we need to change the system according to user needs, we will be able to change it easy and fast. However, there are also some adverse features due to abstractions. To elaborate this situation, as an example some subclasses have some methods and attributes which are not completely necessary. It leads system to allocate unnecessary memory allocation which may affect the performance of the game.

Development Time vs User Experience:

During the design phase of our system, we decided to implement interface components and graphics with external library (Slick2d) as it allows us to smooth performance and adequate resolution. As we decided to use external library, first we consulted our supervisor and got approval. Beside commonly used java libraries (Fx and Swing) we chose Slick2d as we think that it allows us to focus abstraction and OOP design more than struggling with the user interface. As Slick provides better graphics and resolution, users will have smooth game experience.

4.2 Packages

4.2.1 org.newdawn.slick.command

This package provides abstract input by mapping physical device inputs (mouse, keyboard and controllers) to abstract commands that are relevant to a particular game.

4.2.2 org.newdawn.slick.gui

This package includes some extremely simple GUI elements which should be used where a game does not require a full GUI

4.2.3 org.newdawn.slick.tiled

This package contains utilities for working with the tiled utility for creating tiled maps.

4.2.4 org.newdawn.slick.state

This package is for creating state based games which allow the game to be broken down into the different activities the player may take part in, for instance menu, highscores, play and credits.

4.2.5 org.newdawn.slick.loading

This package adds support for deferring loading of resources to a set time to allow loading/progress bar style effects.

4.3 Class Interfaces

4.3.1 InputListener

This interface will be invoked whenever an action occurs with receiving the action events. A listener that will be notified of keyboard, mouse and controller events

4.3.2 KeyListener

This interface will be invoked whenever a key is typed, pressed or released by the user with receiving keyboard events. Describes classes capable of responding to key presses

4.3.3 MouseListener

This interface will be invoked whenever a mouse action is received from the user in order to track the mouse commands. Description of classes that respond to mouse related input events

4.3.4 ControllerListener

This interface will be invoked whenever Notification that a button control has been pressed on the controller. Description of classes capable of responding to controller events

5. References & Glossary

MVC: Model View Controller

System: our game " War of Domination "

Slick2d: http://slick.ninjacave.com/

Façade: design pattern https://www.tutorialspoint.com/design_pattern/facade_pattern.htm

Bruegge, B., & Dutoit, A. H. (2014). Object-oriented software engineering: using UML, patterns, and Java. Harlow, Essex: Pearson.

https://www.visual-paradigm.com/