Bilkent University

Department of Computer Engineering

CS319 - OBJECT-ORIENTED SOFTWARE ENGINEERING

Design Report

Group 1E : Tank Zone

Alperen Koca

Aybüke Ceren Duran

Hakan Türkmenoğlu

Mert Sezer

Supervisor: Eray Tüzün

1. INTRODUCTION

1.1 Purpose of the System

'Tank Zone' is an action and fast-paced game which aims to enjoy the users. It will be improved version of *'diep.io'* game with extra features. Compared to *diep.io'* game, the user will have more fun while playing *'Tank Zone'* game because '*Tank Zone'* game provides game modes such as climate mode, difficulty mode and team mode, difficulty levels, sound, different power-ups and high-quality tank classes. By these extra features, '*Tank Zone'* game will address different users to make them enjoy and teach the strategies of the war between tanks.

1.2 Design Goals

The main goal for a computer game is to make the player entertained. To achieve this important goal, we need to focus on the little details which are unnecessarily directly noticeable at a first glance.

This part of the report provides the design goals of the system such as end user criteria, maintenance criteria, performance criteria and the trade-offs that come with our chosen way of implementation.

a-) End User Criteria

- **Target User Base:** For recent years, other successful games such as '*Farm Heroes Saga'* have the feature of target user base. *'Farm Heroes Saga'* is played by all range of users.

  As emphasized above, one the main goal of the *'Tank Zone'* game is to address different users. At the heart of game world, some players focus on learning game and its strategies in the easiest level while other players like the feeling of victory with having hardness. Therefore, *'Tank Zone'* game provides game difficulty mode for its players.

- **Ease of use:** For the computer games, it is important to use common controls to increase the ease of use. Menu navigations can be controlled by the mouse pointer. While the gamer is playing *'Tank Zone'* game, s/he will be able to move the tank in any direction with using 4 keyboard keys, and s/he will be able to decide where to take aim at while shooting by using mouse pointer.
- **Ease of learning:** When the user plays the game for the first time, s/he will not understand how '*Tank Zone'* game works. So, it is crucial for providing the gamer with a smooth learning tip. 'Help' button on the menu navigator stands for this purpose. In addition, by choosing easy-game mode, the gamer can learn the strategies of the war between tanks. After these simple levels, the player will be pushed towards greater challenges to increase the fun factor.

b-) Maintenance Criteria

- **Extensibility:** Maintaining adding and removing feature is crucial for the lifecycle of a software. We give priority to determine what the game will be and what it will look like. Because of that, the tasks that our system should perform change easily with every new idea. After the launch of '*Tank Zone'* game, there may be possibility that improved version of *'Tank Zone'* might developed. So, '*Tank Zone'* game must be modifiable and we should be able to make changes in the software readily for our future systems.
- **Portability:** *'Tank Zone'* game will be Java-based game. Therefore, *'Tank Zone'* game will work in JRE installed platforms.

- **Readability:** Developing a video game is a complex task. Because of this many games start readable but projects without readability as a goal in mind end up in a spaghetti code. For this reason we aim to use as much as design patterns as possible since they are proven to work.

c-) Performance Criteria

- **Response Time**: Inputs from the keyboard while playing '*Tank Zone'* game is important for moving player's tank because players may be frustrated if their requests are not responded. *'Tank Zone'* game will respond player's actions immediately while also displaying animations and environment of the war between tanks. Therefore we aim to have an FPS of at least 30.
- **Memory:** The game should have reasonable bounds for its RAM and VRAM needs so that it can run on at most 1024 MBs of VRAM and RAM.

d-) Cost criteria:

- **Deployment cost:** The game will be a freeware since it's mainly made for fun.

- **Development cost:** Since the game is not a commercial product, it should be as cheap as possible to develop. This is because we won't have financial security provided by the sales unlike commercial products.

e-) Trade-offs

- **Portability -Performance & Memory: :** 'Tank Zone' game will be designed via Java programming language. We chose Java programming language due to famous for its effectiveness to produce binaries which can run on any processor architecture. However, this situation leads to problem that in all the applications being run in an interpreter called the Java Virtual Machine. When interpreting another perspective, Java will decrease the development time of our project due to having wide range of libraries available. However this has a downside of having higher memory and CPU requirements than using a low-level language like C/C++/Rust. But still we are in favor of this kind of tradeoff.

2. SOFTWARE ARCHITECTURE

2.1 Overview

In this part, we will provide information about the organization of '*Tank Zone'* game in terms of higher level. Therefore, '*Tank Zone'* game will have subsystems. This will produce more maintainable and simpler parts coupled to each other instead of using a huge chunk of code. In *'Tank Zone'* game, we will be using the Model View Controller design pattern because it is appropriate for games like ours.

2.2 System Decomposition

In this part, smaller subsystems of *'Tank Zone'* game will be explained. While designing the subsystems of our game, extensibility and modifiability of our system are taken into consideration.

'Tank Zone' game consists of 3 subsystems:

- View Management
- Game Logic
- Object of Game Management

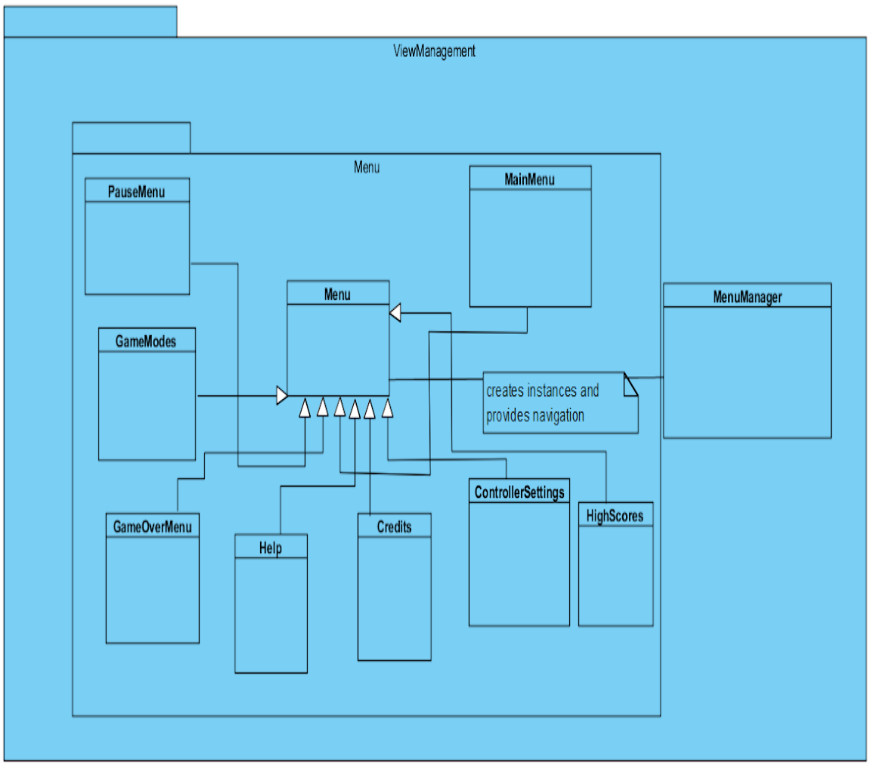


Figure-01: View Management subsystem

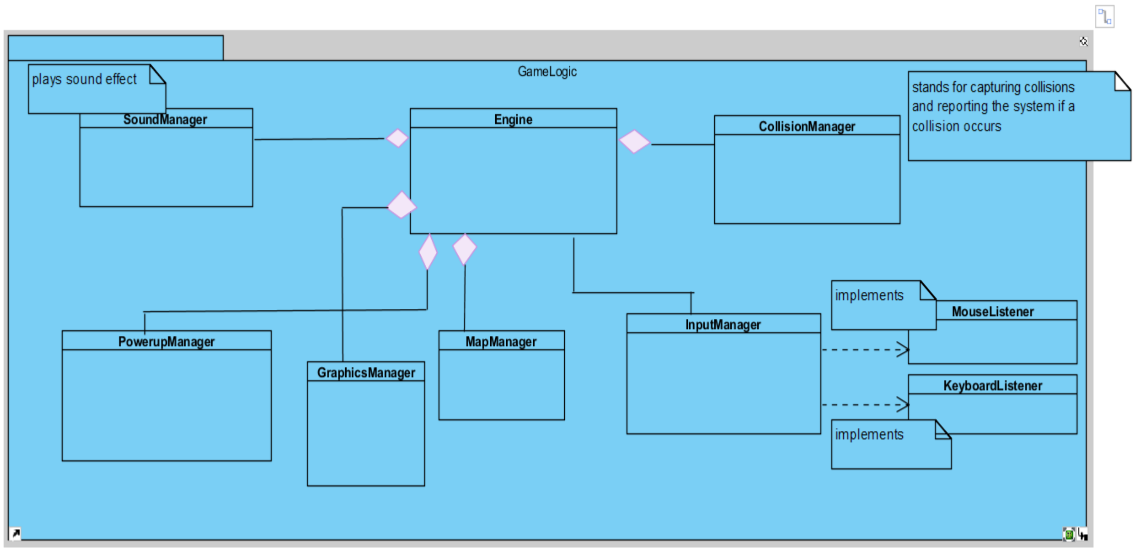


Figure-02: Game Logic subsystem

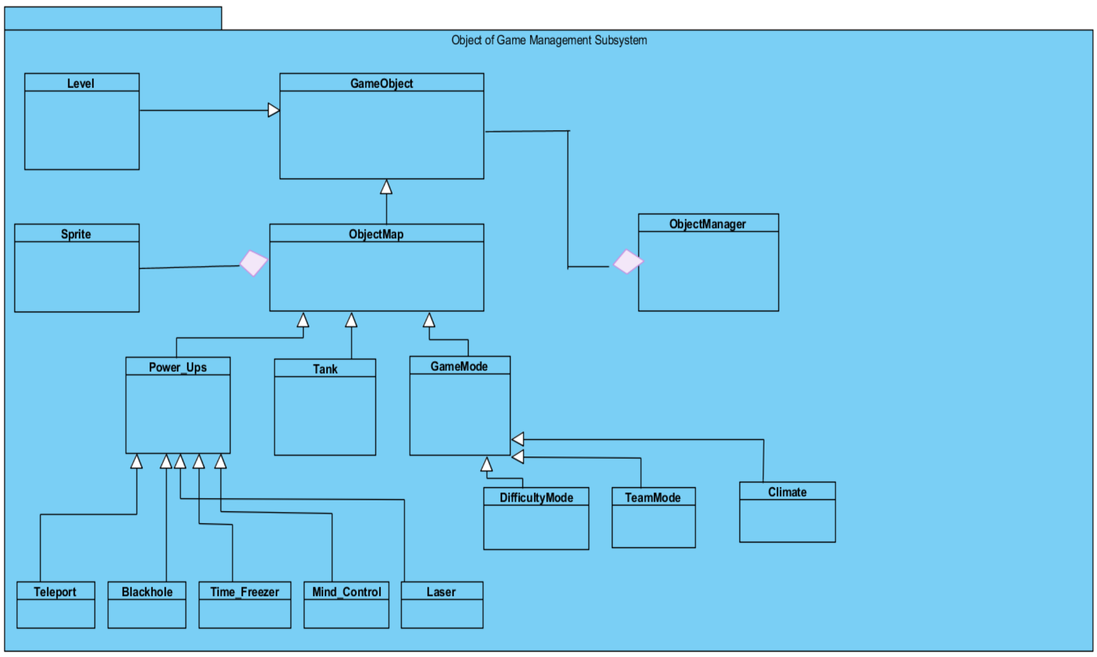


Figure-03: Object of Game Management System

2.3 Architectural Styles

2.3.1 Layers:

In organization of the subsystems of '*Tank Zone'* game, we have organized our system into 3 layers. The user interface is put layer at the top which manages the layers below. View Management layer can interact with these layers to provide required functionality.

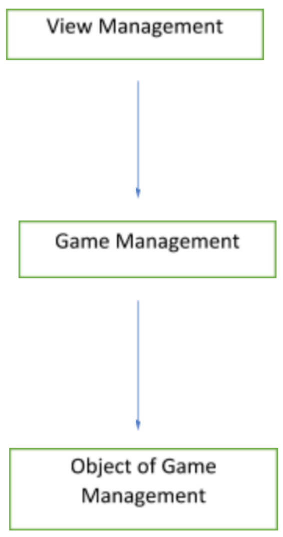


Figure-02: layers of *'Tank Zone'* game

In our 3-layered architecture system we have layers called Presentation, Business Logic layer and Data layer. For the presentation layer, we will have units that interacts with the user. These units request data from business logic layer in order to model it for the user. At the same time the business logic layer implements and updates the data acquired from the data layer. Data layer stores the basic attributes for the units. Those units represent the real life objects. For instance, a tank object is consisted of its graphical presentation (sprites), its physics calculations (business Logic) and finally it's coordinates (data).

2.4 Hardware/Software Mapping

'*Tank Zone'* game will be implemented by the Java programming language. Java 10 will be used for implementation since it has useful features for the development of the game. For the hardware feature of our game '*Tank Zone'* requires a basic mouse and keyboard.

With these hardware and software requirements, any player who uses a desktop computer running Windows/macOS/Linux and Java installed can play '*Tank Zone'* game.

2.5 Persistent Data Management

In the *'Tank Zone'* game, the number of passed levels and the number of upgraded tanks will

be recorded in a comma separated file. This comma separated file will help us in the development cycle.

2.6 Access Control and Security

'Tank Zone' game requires some security managements because different users cannot have same username or password in the team mode. All passwords should be at least 7 characters length and they must include at least an uppercase letter, a lowercase letter and a number. If multiple users are using the same device to play the '*Tank Zone'* game, they have to enter their own username and password for entering their own accounts.

2.7 Boundary Conditions

2.7.1 Execution

'*Tank Zone'* requires Java Runtime Environment installation on the players' computers.

2.7.2 First Run

'*Tank Zone'* game does not have the feature of the saved game state. Therefore, the user

will have only chance to play game.

2.7.3 Termination

In the '*Tank Zone'* game, there will be option to exit '*Tank Zone'* game. Exit Game Button

3. SUBSYSTEM SERVICES

We have divided our system into 3 subsystems to have a better understanding of the whole package. The diagram below shows the main subsystems:

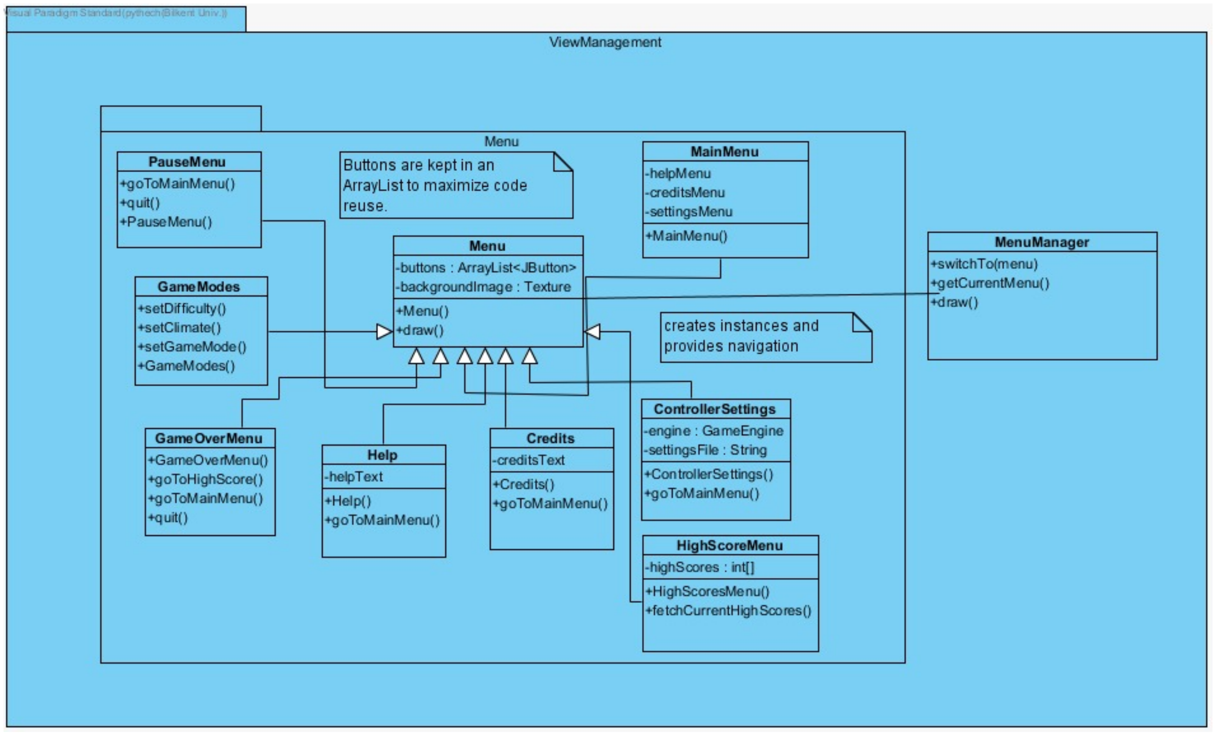
3.1 VIEW MANAGEMENT

MenuManager

Acts as a screen manager for our game. It provides navigation. Since it uses Composite design pattern, we can actually insert as much as menus as we want. This way we can insert a PauseMenu inside the GameFrame without disturbing the overall view system.

Methods:

public void switchTo(Menu menu): Removes the current menu and stats drawing the given parameter.



public Menu getCurrentMenu(): returns the currrent menu

public void draw(): Draws the current menu

Menu

Provides a base implementation for all menus in order to reduce code duplication.

Attributes:
protected ArrayList<JButton> buttons: Stores the list of buttons that will be drawn for the menu. These buttons will be created and have listeners attached in the constructor.

protected Texture backgroundImage: Each menu will have a different background image to be drawn.

Methods:

public void draw(): Draws the buttons and the background image.

MainMenu

Attributes:

private Menu helpMenu: Stores the help menu to switch

private Menu creditsMenu: Stores the credits menu to switch

private Menu settingsMenu: Stores the settings menu to switch

ControllerSettings

A menu to change up, down, left and right keys.

Attributes:

private GameEngine engine : Holds a reference to the engine in order to make changes to the settings.

private String settingsFile : Holds the absolute path for the settings file.

Methods:

public void goToMainMenu(): Switches current view to main menu
HighScoreMenu

Displays the high scores taken from the internet. High scores will be automatically uploaded when a user gets a score in the top 50. This view just displays current high scores, it has no ability to submit.

Attribitues:

private int[] highScores : Holds the current online high scores fetched by the server.

Methods:

public void fetchCurrentHighScores(): Downloads the current high score list from the webserver.

Help

Provides user friendly instructions as to how to play the game.

Attributes:

private String helpText: Stores the text to be shown to the user.

Methods:

public void goToMainMenu(): Switches current view to main menu

Credits

Displays credits.

Attributes:

private String creditsText: Stores the text to be shown to the user.

Methods:

public void goToMainMenu(): Switches current view to main menu

GameModes

Before selecting a new game the user is welcomed with this menu. The users will have option to choose difficulty, climate mode and game mode.

Methods:

public void setDifficulty(): Sets the difficulty for the game, particularly from easy, medium to hard.

public void setClimate(): Sets the climate mode for the game, particularly from winter, temperate and desert.

public void setGameMode(): Sets the game mode for the game, particularly from CTF and FFA.

PauseMenu

Shown whenever the user pauses the game.

Methods:

public void goToMainMenu(): Switches current view to main menu

public void quit(): Exits the game and returns to desktop

GameOverMenu

Shown when the game is over either successfully or not.

Methods:

public void goToHighScore(): Switches current menu to high score

public void goToMainMenu(): Switches current menu to main menu

public void quit(): Exits the game and returns to desktop

3.2 GAME LOGIC

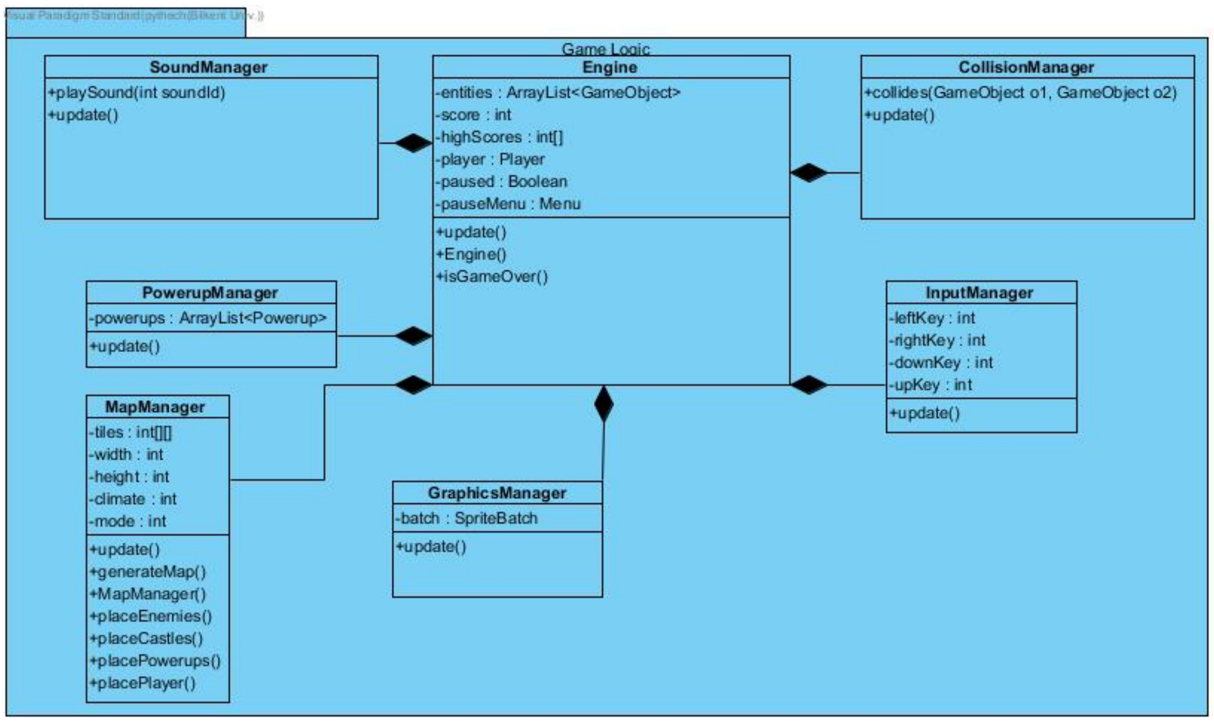
SoundManager

Responsible for handling needs for playing music and sound effects.

Methods:

public void playSound(int soundId): Plays the given sound id.

public void update(): Handles per frame changes.



PowerupManager

Handles the special cases for power up entities.

Attributes:

private ArrayList<Powerup> powerups : Holds the powerup entities.

Methods:

public void update(): Handles per frame changes.

MapManager

Responsible for overall map manager. Draws the background tiles in the map and additionally creates all the entities in the game. Generates a game procedurally so that each map will have a fresh look but since it is procedurally generated we will have our own rules to make the game more interesting than just placing random objects across the map.

Attributes:

private int[][] tiles: Tile types for each tile as a 2D matrix.
private int width : Width of the map.
private int height: Height of the current map.
private int climate : Climate mode of the current map from winter, temperate and desert.
private int mode : Gamemode of the current map from CTF and FFA.

Methods:

public void update() : Per frame update for the current map
public void generateMap() : Generates a procedural with a seed taken from a PRNG.
public void placeEnemies(int seed) : Generates enemy locations using the seed
public void placeCastles(int seed) : Generates castle locations using the seed
public void placePowerups(int seed) : Generates powerup locations using the seed
public void placePlayer(int seed) : Generates the player location using the seed

GraphicsManager

Responsible for drawing the entities inside the game.

Attributes:

private SpriteBatch batch : The class responsible for drawing textures using OpenGL in a batched fasion.

Methods:

public void update() : Draws entities for each frame

InputManager

Provides user interaction with the player entity.

Attributes:
private int leftKey : Keymap id for steering the tank to the counter-clockwise side.
private int rightKey : Keymap id for steering the tank to the clockwise side.
private int upKey : Keymap id for increasing the tank's velocity.
private int downKey : Keymap id for decreasing the tank's velocity.

Methods:

public void update() : Checks for incoming inputs and sends commands if necessary to other parts of the system.

CollisionManager
Detects cases for collisions and handles accordingly.
Methods:
public bool collides(GameObject o1, GameObject o2): Checks if given two objects collide using simple Circle-Circle collision algorithm for increased performance.
public void updates(): Checks if the objects collide

Engine

Main Façade class for managing the game by splitting up responsibility to several classes.

Attributes:
private ArrayList<GameObject> entities : Every game object ever created will reside in here.
private int score : The current score achieved by the user.
private Player player : The current player class that he user controls.

private bool paused : The engine stops updating the managers if the user decides to pause the game.
private Menu pauseMenu : Pause menu is exceptionally handled in case the game is paused.

Methods:
public void update() : Updates every manager class.
public bool isGameOver() : Checks conditions for the ending, e.g tank has no ammo left, health left etc.