Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software Engineering

Design Report

**Man of War**

**Group 1-D**

* Berat Tuna Karlı
* Ceren Uysal
* Serdar Erkal
* Selim Fırat Yılmaz

Contents

[How implementation went? 3](#_Toc497602976)

[Are there any major changes to the design had to be made due to complications during the implementation? 4](#_Toc497602977)

[User’s Guide 4](#_Toc497602978)

[The status of the implementation 8](#_Toc497602979)

[Exposed Documentation of the software system (Appendix) 9](#_Toc497602980)

[Class GameStateManager 10](#_Toc497602981)

[Class GameThread 11](#_Toc497602982)

[Class LevelFactory 11](#_Toc497602983)

[Class Block 12](#_Toc497602984)

[Class Character 12](#_Toc497602985)

[Class CharacterGraphicsComponent 13](#_Toc497602986)

[Class CharacterInputComponent 13](#_Toc497602987)

[Class CharacterPhysicsComponent 14](#_Toc497602988)

[Class Door 14](#_Toc497602989)

[Class DoorPhysicsComponent 14](#_Toc497602990)

[Class GameObject 15](#_Toc497602991)

[Class Inventory 15](#_Toc497602992)

[Class Velocity 15](#_Toc497602993)

[Enum Difficulty 16](#_Toc497602994)

[Class Item 17](#_Toc497602995)

[Class ItemGraphicsComponent 17](#_Toc497602996)

[Class ItemPhysicsComponent 18](#_Toc497602997)

[Enum ItemType 18](#_Toc497602998)

[Class Mob 18](#_Toc497602999)

[Class MobGraphicsComponent 19](#_Toc497603000)

[Class ButtonListener 19](#_Toc497603001)

[Class GamePanel 20](#_Toc497603002)

[Class FileManager 20](#_Toc497603003)

[Class GraphicsManager 20](#_Toc497603004)

[Class InputManager 21](#_Toc497603005)

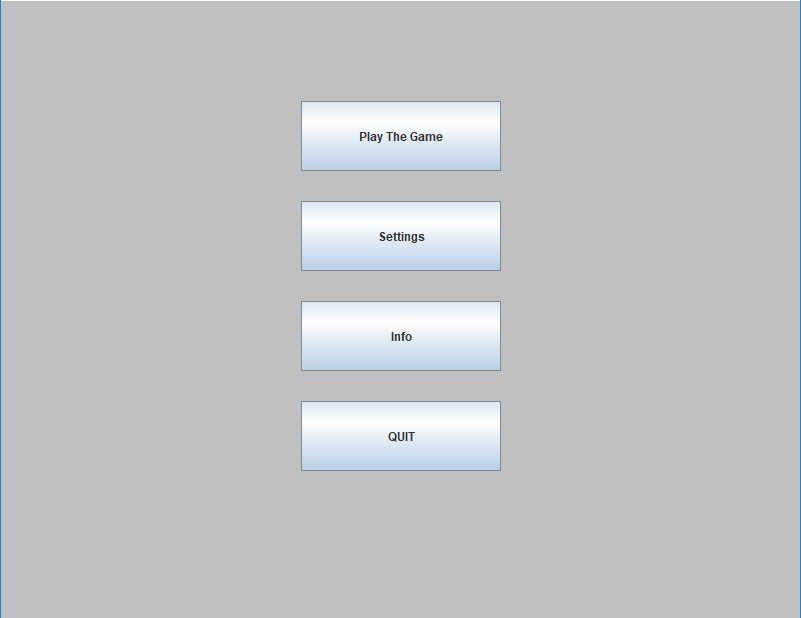
# How implementation went?

The implementation process was a lot easier than we have expected. Since we have prepared very detailed and useful analysis and design reports. We have made use of the diagrams such as the class diagram, the sequence diagrams, use case diagrams and so forth. We have some difficulties during the implementation. The integration and the exacution of the code was kind of hard, because we have implemented our codes independently.

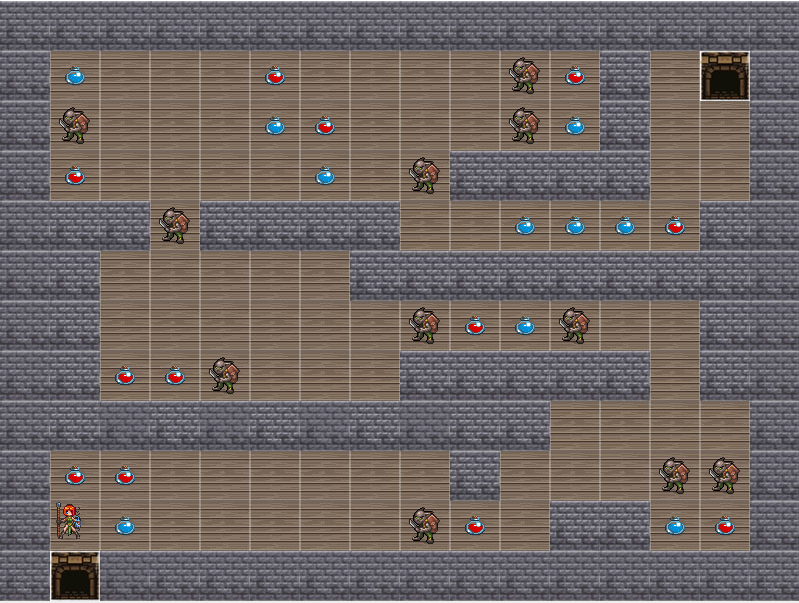
# Are there any major changes to the design had to be made due to complications during the implementation?

There are two major changes in order to handle the implementation. The movement of the main character with the help of Thread and the conversion of  items into the rectangles. We have used Threads in order to update the whole system by providing graphical smoothess of the game. We represented bounding boxes of the objects via rectangle in order to resolve collusions easily.

# User’s Guide

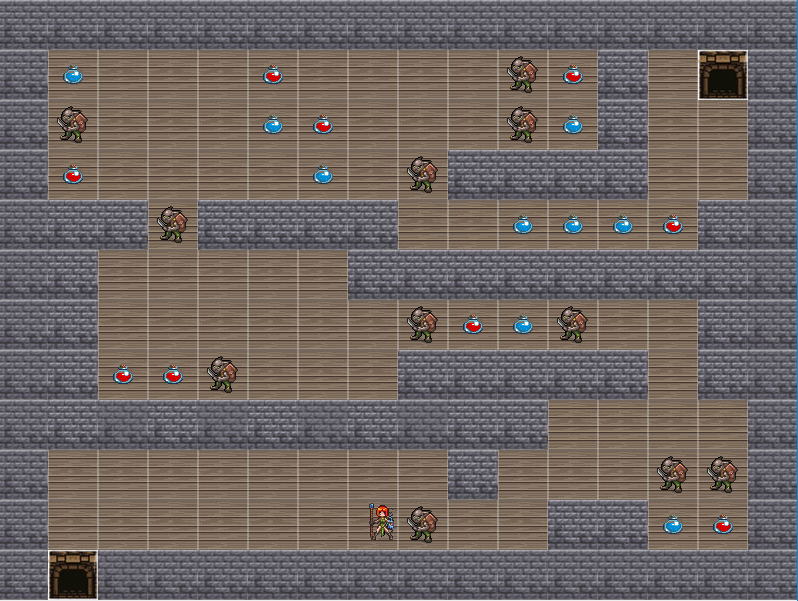


There is a main menu which includes 4 sections. These 4 sections are Play The Game, Settings, Info, Quit. If the player wants to play the game (s)he should click on the Play Game button.

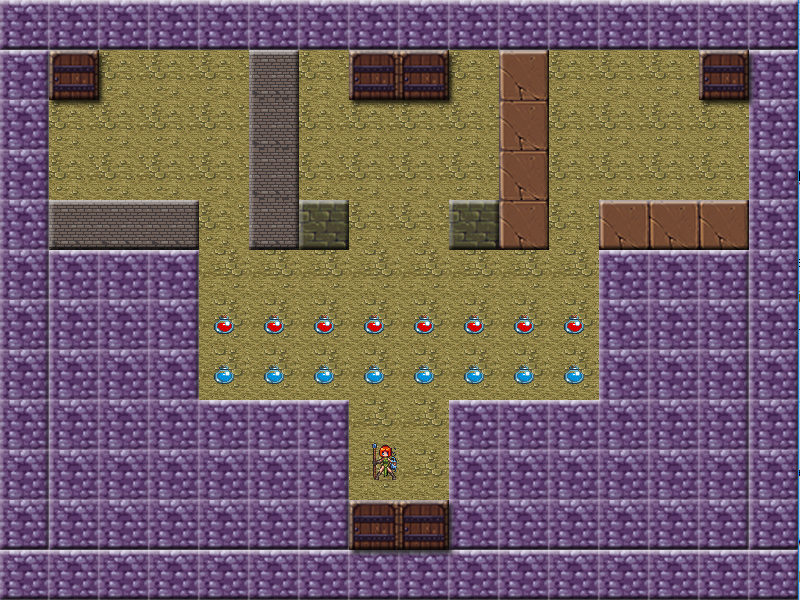
****

(Level 1)

After the click, the game map is uploaded and (s)he directly starts the game from the first level. There will be red and blue potions to be collected to increase the red power and blue power of the character. There will be some mobs to fight and defeat in order to reach the gate and pass to the next level. In addition to this, when character defeats the mobs, (s)he gains extra power.

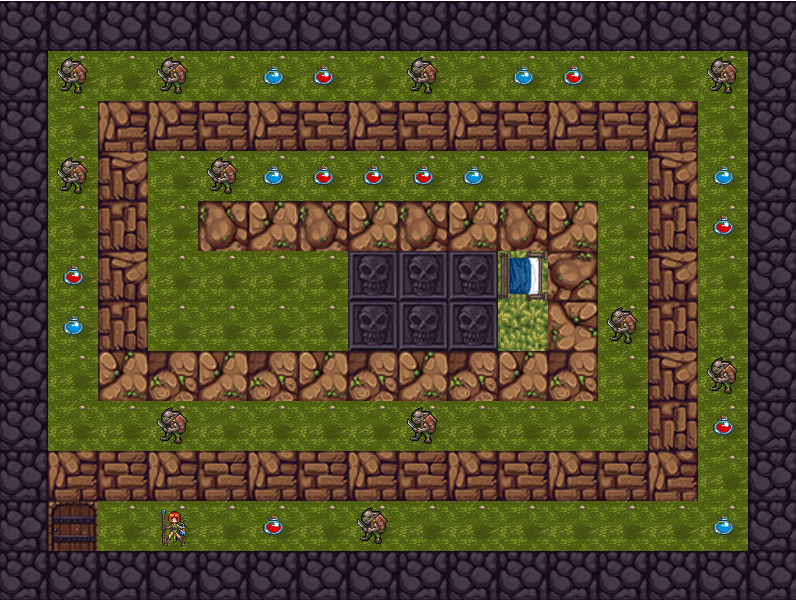
****

Character moves with the help of key arrows (four directions: left, right, up, down). When character encounters with a mob, the pop-up screen will appear and ask user to enter required commands in a limited time.

****

(Level 2)

Different doors are opening to different levels. One of the doors is responsible for tricking the player, it opens to the same level.

****

(Level 3)

Level 3 is the last and the hardest level of the game. The purpose is reaching the bed(next to skulls) to sleep a while.

The process of opening the game is very easy. If the player downloads the jar file and clicks on it (s)he directly starts playing the game.

# The status of the implementation

There are some parts which are implemented with some missing functionalities or not implemented. The part with some missing needed functionalities is the quick time event of the battle system. The missing functionalities are not closing the pop-up frame when the player waits too much (7 s) without entering the required key arrows and the pop-up frame does not ask player the required keys again if the player enters any wrong key value.

One of the unimplemented part is the settings which includes making changes of the volume (up or down), the full-screen option and the selecting the difficulty of the current level (easy, normal or hard). We have the class which is related to difficulty but we have not set the related constant variables, therefore we have not involved this part in to the game itself. The sound related classes have not been implemented at all. Another unimplemented part is the info part. We have the information button but it does not contain any content. We have not display the health bar and power of the main character on the game screen.

# Exposed Documentation of the software system (Appendix)

* **Config class contains the constants of the game. This class provides an easy interface to change and play with important constants of the game.**

|  |  |
| --- | --- |
| **Modifier and Type** | **Field and Description** |
| **static long** | **DELAY**  **The delay between each iteration of game loop.** |
| **static int** | **TILE\_HEIGHT**  **The height of each tile in game.** |
| **static int** | **TILE\_WIDTH**  **The width of each tile in game.** |
| **static double** | **VELOCITY**  **The velocity of the character.** |

## Class GameStateManager

**GameStateManager is to control the game state and contains all its game objects. Manages coherence between utility managers and game objects. Contains all managers and game objects but interferes with them as little as possible.**

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| **java.util.List<Block>** | **getBlocks()**  **Returns the blocks in the game** |
| **Character** | **getCharacter()**  **Returns the character** |
| **java.util.List<Door>** | **getDoors()**  **Returns the doors in the game** |
| **FileManager** | **getFileManager()**  **Returns the composited FileManager instance** |
| **GraphicsManager** | **getGraphicsManager()**  **Returns the composited GraphicsManager instance** |
| **InputManager** | **getInputManager()**  **Returns the composited InputManager instance** |
| **java.util.List<Mob>** | **getMobs()**  **Returns the mobs in the game** |
| **java.util.List<Item>** | **getNonTakenItems()**  **Returns the non-taken items on the map** |
| **double** | **getPassedSeconds()**  **Returns the passed seconds since the last iteration** |
| **double** | **getTotalPassedSeconds()**  **Returns the passed seconds since the beginning of the game** |
| **void** | **initLevel(int levelNum)**  **Inits level and shows it on screen.** |
| **void** | **startGame()**  **Starts the game with level 1** |
| **void** | **update(double passedSeconds)**  **Executes game objects' update methods.** |

## Class GameThread

* **GameThread class is the responsible of the game loop. Executes update methods of the game objects through gamestatemanager and repaints the GamePanel.**

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| **void** | **run()**  **The method to be runned when thread starts.** |

## Class LevelFactory

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| **Block[]** | **createBlocks(int levelNum)** |
| **Character** | **createCharacter(int levelNum)** |
| **Door[]** | **createDoors(int levelNum)** |
| **Mob[]** | **createMobs(int levelNum)** |
| **Item[]** | **createNonTakenItems(int levelNum)** |

## Class Block

Block is the game object that represents restricted tile in the map

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Class Character

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| int | **getBluePower**()  Returns the blue power |
| int | **getHealth**()  Returns the health value |
| **Inventory** | **getInventory**()  Returns the inventory of character |
| int | **getMaxHealth**()  Returns the max health value |
| int | **getRedPower**()  Returns the red power value |
| **Velocity** | **getVelocity**()  Returns the velocity of character |
| void | **setBluePower**(int bluePower)  Changes the blue power value |
| void | **setHealth**(int health)  Changes the current health value |
| void | **setMaxHealth**(int maxHealth)  Changes the max health value |
| void | **setRedPower**(int redPower)  Changes the red power value |
| void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Class CharacterGraphicsComponent

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GraphicsManager** graphicsManager)  The method that is executed each iteration of game loop. |

## Class CharacterInputComponent

CharacterInputComponent is the responsible class for the input related business of character. Works in coherence with InputManager.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**InputManager** inputManager)  The method that is executed each iteration of game loop. |

## Class CharacterPhysicsComponent

The physics component of the character. Works in coherence with other game objects' physics components.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Class Door

Door is the game object that represents the situation when user is in, level changes.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| int | **getTargetLevelNum**()  Returns the target level number |
| void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Class DoorPhysicsComponent

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GameStateManager** gameStateManager) |

## Class GameObject

* GameObject class is a main template of the objects in the game. Common properties of objects in the game are represented in this class.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| java.awt.Rectangle | **getBoundingBox**()  Returns the bounding box of the game object. |
| abstract void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Class Inventory

* Inventory class is a convenient name contains items with restricted access. That is just unmodifiable list of items.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| java.util.List<**Item**> | **getItems**()  Returns the list of items |



## Class Velocity

|  |
| --- |
| **Constructor and Description** |
| **Velocity**()  A constructor that initiates x and y components of the velocity to 0.0 |
| **Velocity**(double xVelocity, double yVelocity)  Simply, constructor |

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| double | **getX**()  Returns the X component of velocity |
| double | **getY**()  Returns the Y component of velocity |
| void | **setX**(double xVelocity)  Changes the value of X component of velocity |
| void | **setY**(double yVelocity)  Changes the value of Y component of velocity |

## Enum Difficulty

* Contains the difficulties of the game as EASY, NORMAL, HARD.

|  |
| --- |
| **Enum Constant and Description** |
| **EASY** |
| **HARD** |
| **NORMAL** |

## Class Item

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| int | **getBonusBluePower**()  Returns the bonus blue powerto be given when the item is picked |
| int | **getBonusHealth**()  Returns the bonus health to be given when the item is picked |
| int | **getBonusRedPower**()  Returns the bonus red power to be given when the item is picked |
| **ItemType** | **getItemType**()  Returns the type of the item |
| boolean | **isVisible**()  Returns whether the item is visible on the map |
| void | **setVisible**(boolean visible)  Changes visibility of the item |
| void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Class ItemGraphicsComponent

* ItemGraphicsComponent is the responsible class for the graphics related business of character. In other words, draws the item in the required position. Works in coherence with GraphicsManager.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GraphicsManager** graphicsManager)  The method that is executed each iteration of game loop. |

## Class ItemPhysicsComponent

* The physics component of the item. Works in coherence with other game objects' physics components.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Enum ItemType

* ItemType enum is to represent various items like red potion

|  |
| --- |
| **Enum Constant and Description** |
| **BLUE\_POTION** |
| **RED\_POTION** |

## Class Mob

Represents mob game object

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GameStateManager** gameStateManager)  The method that is executed each iteration of game loop. |

## Class MobGraphicsComponent

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **update**(**GraphicsManager** graphicsManager) |

## Class ButtonListener

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| java.awt.event.ActionListener | **infoButtonListener**(javax.swing.JFrame frame)  creates new info panel to frame and removes main menu panel from frame |
| java.awt.event.ActionListener | **playButtonListener**(javax.swing.JFrame frame)  creates new game panel to frame and removes main menu panel from frame |
| java.awt.event.ActionListener | **quitButtonListener**(javax.swing.JFrame frame)  quit from game |
| java.awt.event.ActionListener | **settingsButtonListener**(javax.swing.JFrame frame)  creates new settings panel to frame and removes main menu panel from frame |
| java.awt.event.ActionListener | **turnFromInfoListener**(javax.swing.JFrame frame)  creates new main menu panel to frame and removes info panel from frame |
| java.awt.event.ActionListener | **turnFromSettingsListener**(javax.swing.JFrame frame)  creates new main menu panel to frame and removes settings panel from frame |

## Class GamePanel

|  |  |
| --- | --- |
| void | **createGamePanel**(javax.swing.JFrame frame)  creates the game panel and starts the game |

## Class FileManager

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| java.awt.image.BufferedImage | **getImage**(java.lang.String urlString)  Returns the requested image as BufferedImage |
| java.awt.image.BufferedImage | **getImage**(java.net.URL url)  Returns the requested image as BufferedImage |

## Class GraphicsManager

GraphicsManager class is responsible for drawing the game screen.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | **draw**(java.awt.image.BufferedImage bufferedImage, java.awt.Rectangle boundingBox) |
| java.awt.image.BufferedImage | **getFullImage**()  Returns the full image to be drawn on GamePanel |
| java.awt.image.BufferedImage | **getResource**(java.lang.String name) |
| void | **resetFullImage**()  Resets full image by creating |
| void | **setFullImage**(java.awt.image.BufferedImage fullImage) |

## Class InputManager

* InputManager is responsible for tracking user inputs and stores them.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| java.util.Map<java.lang.Integer,java.lang.Boolean> | **getKeys**()  Returns the statuses of keys (whether pressed or not pressed) |
| boolean | **isPressed**(int keyEvent)  Returns whether the given key is pressed. |
| void | **keyPressed**(java.awt.event.KeyEvent e)  Invoked when a key has been pressed. |
| void | **keyReleased**(java.awt.event.KeyEvent e)  Invoked when a key has been released. |
| void | **keyTyped**(java.awt.event.KeyEvent e)  Unused method but implemented due to the KeyListener interface. |