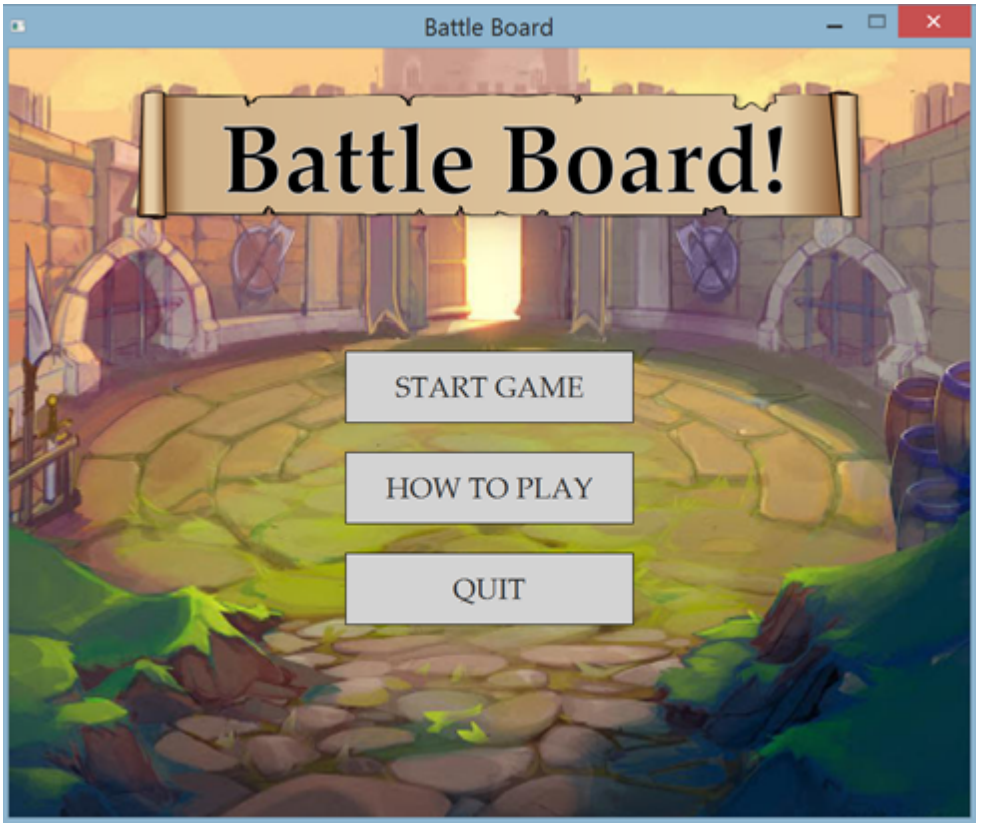
Battle Board! Documentation



Created by

Chayakorn Vongbunsin 6331307321

Phitchaya Charansathianchai 6331330721

2110215 Programming Methodology

Semester 2 Year 2020

Chulalongkorn University

Battle Board!

Introduction

Battle Board! is inspired from a 2-player strategy game called “chess”. The objective of the game is to defeat all of the enemy’s fighters and win 2 rounds out of 3 rounds.

Rules

Each round, the player will pick 5 fighters from 8 random fighters as shown in figure 1. If the time is up and you are incomplete selection, the game will automatically fill up your team by selecting from the top of the box pane as shown in figure 2.

figure 1

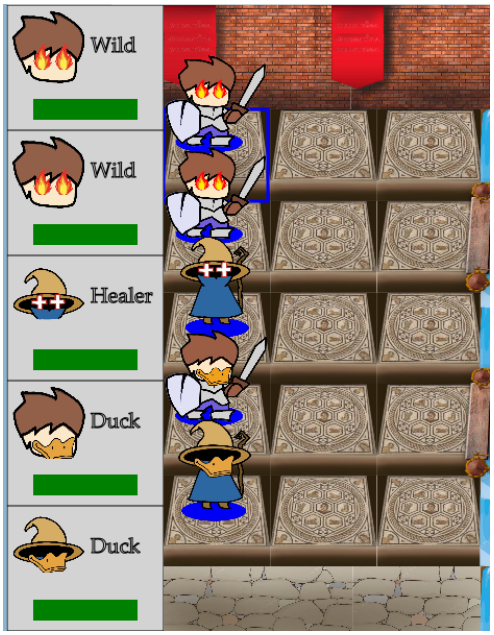


figure 2

Each player will have 9 turns in each round starting from player 1 and switch to player 2 after player 1 ends his/her turn. **Each turn**, a player can use every fighter in his/her team by choosing **one action,** which is “move” or “attack”, or “heal” only for a specific type of fighter, for each fighter which has different abilities or stats depending on the type of fighter as shown in figure 3.

figure 3

Provided that one of the players has no fighter left on his/her team or both players complete 9 turns, the round will end. The winner of each round depends on 1) The player has more fighters left than the opponent. 2) The player has a higher total percentage of fighters' health than the opponent. 3) The round will end up with a tie round.

Fighters

There are five different Fighters: DuckFighter ,HealerFighter, SpeedyFighter, ToughFighter and WildFighter, which each got a special ability or increase in stats, and two attacking types: melee and range. So there are ten in total unique Fighters. Melee Fighters can attack an enemy which is 1 box far away from them (only adjacent boxes), 2 boxes far for ranges. Every Fighters have basic stats depending on attacking types.

Though melee Fighters have disadvantages in attack range, melee Fighters have significantly greater base stats than range Fighters.

Duck Fighter

Duck Fighter has an annoying special ability: dodging attacks, referring to their name “Duck”. Duck Fighter has a 25 to 35 percent chance to dodge an attack. Dodge chance will be generated randomly for each Duck Fighter.

Healer Fighter

A Healer Fighter’s special ability is that he/she can heal their allies or theirself. Healer Fighter’ heal is another action besides move and attack. A player can choose to heal instead of moving or attacking. In addition, Healer Fighter regenerates their Hit Point by 5 percent of their Maximum Hit Point once every end of a turn.

Speedy Fighter

Speedy Fighter is able to move 2 boxes far, while other Fighters can only move 1 box far. This allows them to make great spaces for their team as well as battling advantageously.

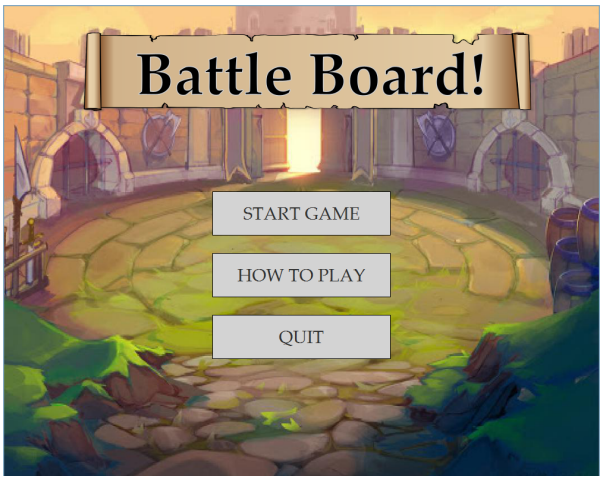
Tough Fighter

Tough Fighter is tough as their name. Tough Fighter has no special ability but their Defend and Hit Point will be greater than normal. Defend could be 0 to 20 percent increased, Hit Point 20 to 50 percent increased, generated randomly. Tough Fighter also regenerates their Hit Point every end of a turn as Healer Fighter, but by 2 percent of Max Hit Point a time.

Wild Fighter

Wild Fighter is wild.Wild Fighter's Attack will be greater than normal. Attack could be 25 to 40 percent increased, generated randomly.

Main Screen



The main screen has 3 buttons including the start game button, the how to play button and quit button.

Start Game button

Clicking on this button will open the pane that will provide you to input your name before starting the game.

Name Pane

In this pane, two players need to fill the name and confirm before clicking start the game. If both player's names are not confirmed, clicking on the start button will do nothing.

The name of the player must be as following:

1) It must contain 1-6 characters.
2) It must contain only English alphabet and number.
3) Both players cannot use the same name.

If you are not filling any name or filling more than 6 characters, the error will alert and clear the name you filled as shown in figure 4 and figure 5.

figure 4

figure 5

If you are not using English alphabet or number, the error will alert and clear the name you filled as shown in figure 6.

If you are using the same name, the error will alert and clear the name you filled as shown in figure 7.

figure 6

figure 7

How To Play Button

Clicking on this button will open a pane that gives you information about this game.

How To Play Pane

In this pane, you can scroll the pane to read more information in it and click the back button to go to the main menu.The information in this pane including fighter description, map, and how to control the fighter.

Quit Button

Clicking on this button will close the program.

Game Screen

This is the game screen where the battle takes place. The battle had two main phases, Pre-Battle phase and Battle phase.

Pre-Battle Phase

This phase has 60 seconds for both players to choose their five Fighters from randomly provided eight for the battle.

Top Pane (Status Pane)

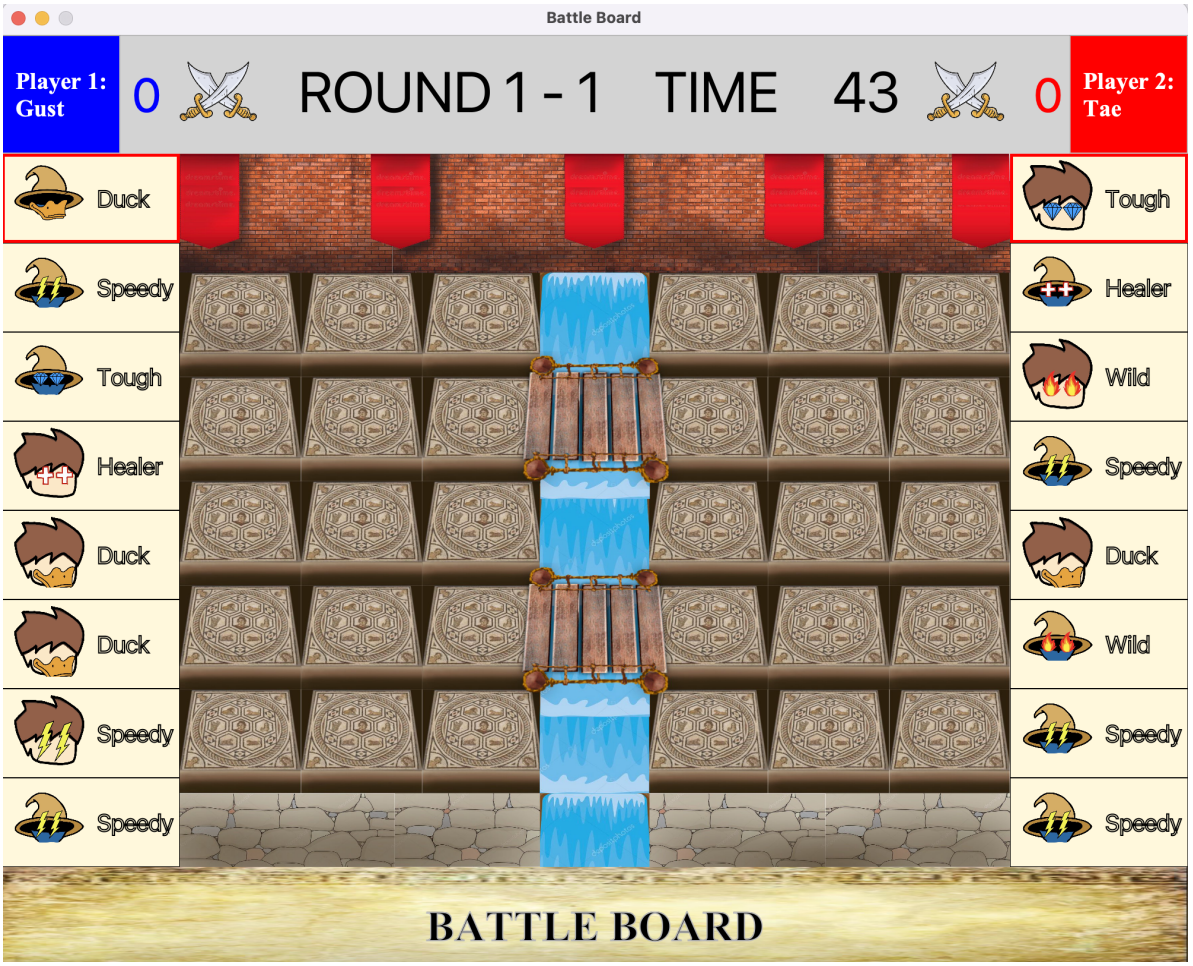
- ❏ Players' names are on the top left and right corner.
- ❏ Beside the names is the round won count for each player. Currently 0 to 0.
- ❏ ROUND tells round count and turn count respectively.
- ❏ TIME is the time remaining to choose your Fighters.

Left and Right Pane (Player Box Pre Battle Pane)

- ❏ Contain randomly provided 8 Fighters.
- ❏ 1) Use W,S to control the pointer and press F to choose for Player1 2) I,K to control the pointer and SEMICOLON (;) to choose for Player2.

Middle Pane (Board Pane)

❏ 5 rows 7 columns of boxes for Fighters to battle with two bridges to cross the river on the fourth column.



Pre Battle Phase

Battle Phase

When both players get their five Fighters, it's time to battle.

Top Pane (Status Pane)

❏ TIME is the time remaining for a player to decide and play their Fighters in each turn. Players got 60 seconds per turn.

❏ Other components are already informed in the Pre Battle Phase.

Left and Right Pane (Player Box Battle Pane)

❏ Contain all Fighters' status for each Player, including Fighter names and their Hit Point.

Middle Pane (Board Pane)

❏ Five chosen Fighters from Pre Battle will be placed on the first columns of each Player.

Battle Phase

Controlling Fighters

Each turn a Player can play all of his/her Fighters. Each Fighter can do one action per turn.

Use W,A,S,D to control the pointer and press F to choose a Fighter (for Player1)(in Figure 1) or I,J,K,L to control the pointer and press SEMICOLON (for Player2) to choose a Fighter. Then the Action Box will show up (in Figure 2).

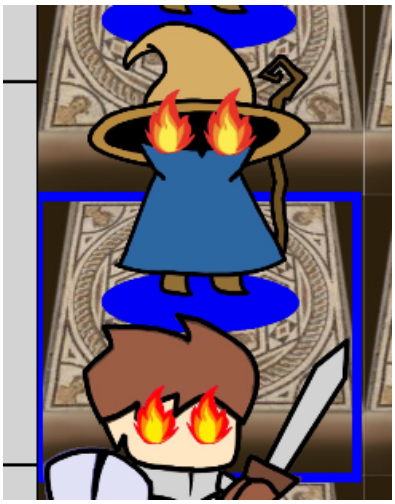


Figure 1.

Figure 2.

Since there is no enemy Fighter in the chosen Fighter's attack range, the only option available is move. (or you can stand still and do nothing by pressing SPACE.

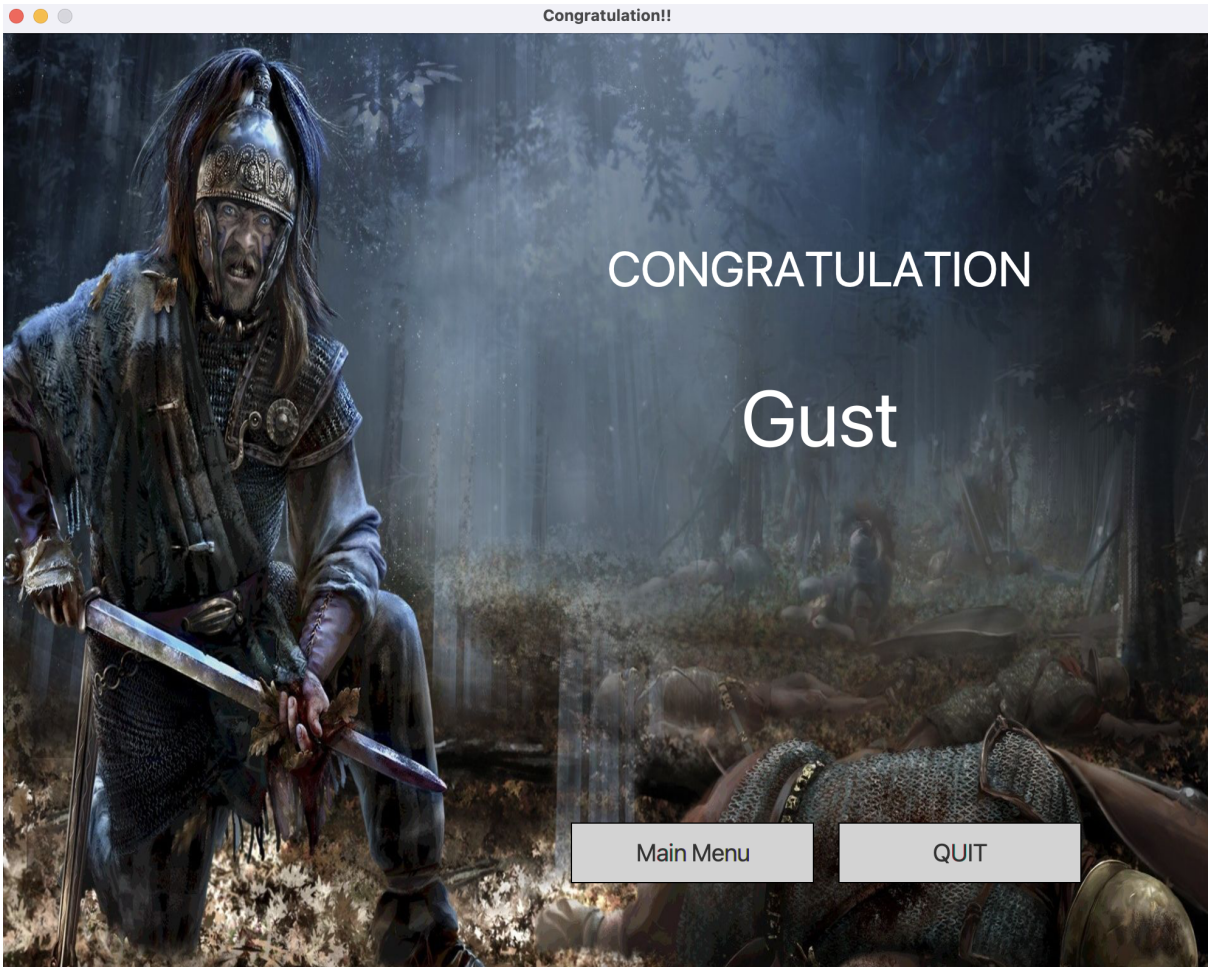
If there is at least one enemy Fighter in attack range, you can choose an enemy in the range to attack.

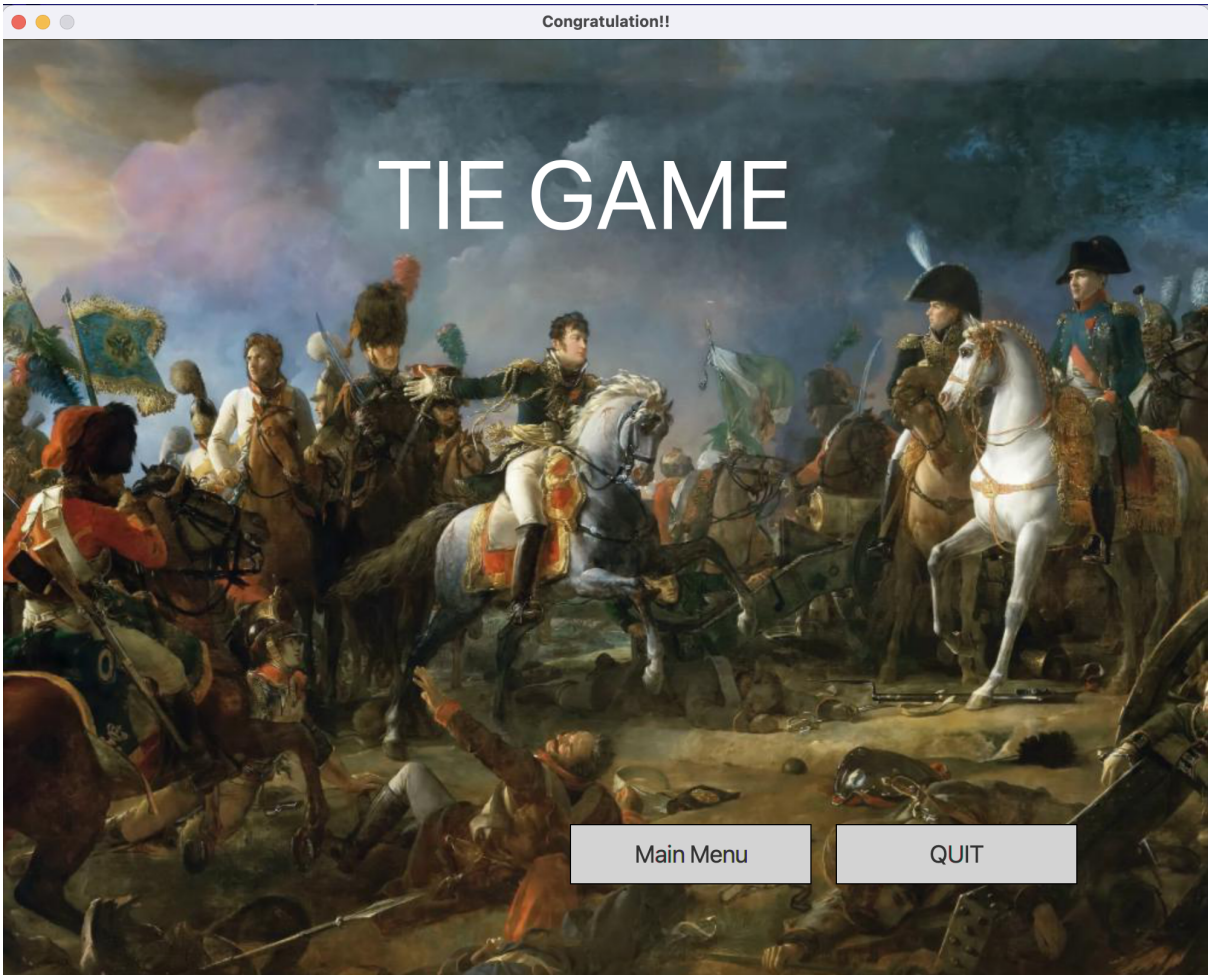
Then the battle goes on…. If one player is out of Fighters they lose and another player gets a score. Or if both players play completely 9 turns each, will check which player got more Fighters alive, if they are the same then check sum of Hit Point by percent of sum of Max Hit Point of every Fighters alive. Else nobody gets a score (This case is a super rare case).

If one player reaches a score of 2, they win. Or if both players have the same score when the final round ends, the game is tied.

End Screen

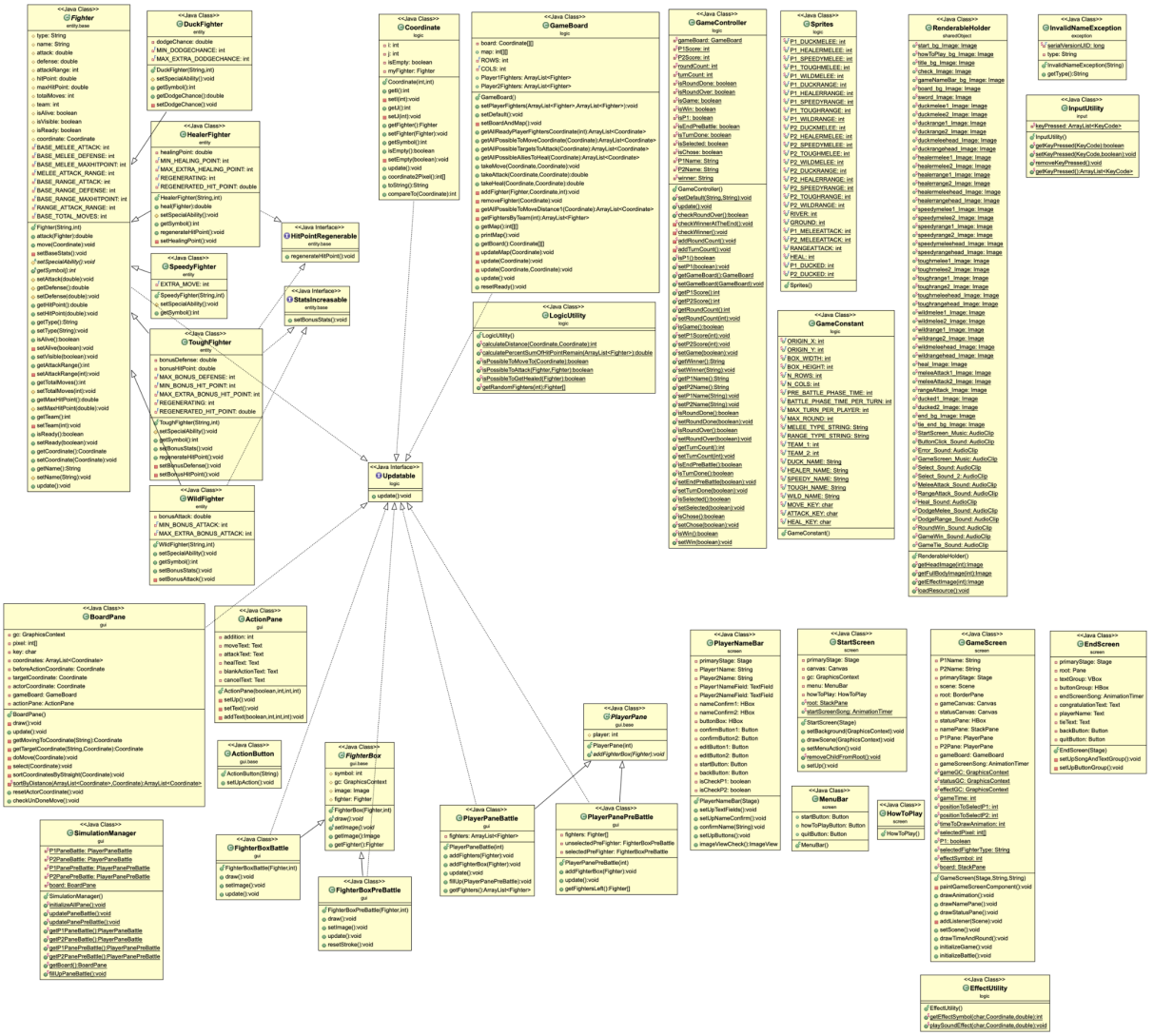
End Screen congratulates the winner, if the game is not tied.





Here you have two options to go to the Main Menu or just quit the game.

Class Diagram



1. Package application

1.1 class **Main** extends Application

1.1.1 Method

Name	Description
+ void main(String[ ] args)	The entry point of the application
+ void start(Stage primaryStage)	- initialize start screen - set stage to unresizable - show stage

2. Package entity

2.1 class **DuckFigher** extends Fighter

DuckFighter have a chance to dodge attacks.

2.1.1 Fields

Name	Description
- double dodgeChance	percent chance to dodge an attack, dodgeChance be in [MIN_DODGECHANCE, MIN_DODGECHANCE + MAX_EXTRA_DODGECHANCE]
- final int MIN_DODGECHANCE	minimum dodge chance possible by percent
- final int MAX_EXTRA_DODGECHANCE	maximum extra dodge chance possible by percent

2.1.2 Constructor

Name	Description
+ DuckFighter(String type, int team)	- call super constructor with parameters - set name to “Duck”

2.1.3 Methods

Name	Description
# void setSpecialAbility()	set dodge chance
+ int getSymbol()	return symbol depending on team and attacking type
+ int getDodgeChance()	return dodgeChance
- void setDodgeChance()	set dodgeChance with MIN_DODGECHANCE and

	MAX_EXTRA_DODGECHANCE

2.2 class HealerFighter extends Fighter implements HitPointRegenerable

HealerFighter can heal an ally or self. Heal is an extra action from Move and Attack.

2.2.1 Fields

Name	Description
- double healingPoint	heal point, be in [MIN_HEALING_POINT, MIN_HEALING_POINT + MAX_EXTRA_HEALING_POINT]
- final int MIN_HEALING_POINT	minimum healing point possible
- final int MAX_EXTRA_HEALING_POINT	maximum extra healing point possible
- final int REGENERATING	percent of max hit point to regenerate hit point
- final double REGENERATED_HIT_POINT	point to regenerate hit point

2.2.2 Constructor

Name	Description
+ HealerFighter(String type, int team)	- call super constructor with parameters - set name to “Healer”
+ double heal(Fighter ally)	increase ally’s hit point by healingPoint, return total point healed
# void setSpecialAbility()	set healing point
+ int getSymbol()	return symbol depending on team and attacking type

+ void regenerateHitPoint()	increase hit point by REGENERATED_HIT_POINT
- void setHealingPoint()	set healing point with MIN_HEALING_POINT and MAX_EXTRA_HEALING_POINT

2.3 class **SpeedyFighter** extends Fighter

SpeedyFighter can move 2 boxes far.

2.3.1 Fields

Name	Description
- final int EXTRA_MOVE	extra move

2.3.2 Constructor

Name	Description
+ SpeedyFighter(String type, int team)	- call super constructor with parameters - set name to “Speedy”

2.3.3 Methods

Name	Description
# void setSpecialAbility()	set new total moves with EXTRA_MOVE
+ int getSymbol()	return symbol depending on team and attacking type

2.4 class **ToughFighter** extends Fighter implements HitPointRegenerable, StatsIncreasable

ToughFighter have defense and hit point boost.

2.4.1 Fields

Name	Description
- double bonusDefense	percent to increase defense
- double bonusHitPoint	percent to increase hit point
- final int MAX_BONUS_DEFENSE	maximum bonus defense possible by percent
- final int MIN_BONUS_HIT_POINT	minimum bonus hit point possible by percent
- final int MAX_EXTRA_BONUS_HIT_POINT	maximum extra bonus hit point possible by percent
- final int REGENERATING	percent of max hit point to regenerate hit point
- final double REGENERATED_HIT_POINT	point to regenerate hit point

2.4.2 Constructor

Name	Description
+ ToughFighter(String type, int team)	- call super constructor with parameters - set name to “Tough” - set bonus stats

2.4.3 Methods

Name	Description
# void setSpecialAbility()	does nothing. ToughFighter have no special ability
+ int getSymbol()	return symbol depending on team and attacking type
+ void setBonusStats()	- set bonus defense and hit point - set new defense and hit point

+ void regenerateHitPoint()	increase hit point by REGENERATED_HIT_POINT
- void setBonusDefense	set bonus defense with MAX_BONUS_DEFENSE
- void setBonusHitPoint	set bonus hit point with MIN_BONUS_HIT_POINT and MAX_EXTRA_BONUS_HIT_POINT

2.5 class WildFighter extends Fighter implements StatsIncreasable

WildFighter have attack boost.

2.5.1 Fields

Name	Description
- double bonusAttack	percent to increase attack
- final int MIN_BONUS_ATTACK	minimum bonus attack possible
- final int MAX_EXTRA_BONUS_ATTACK	maximum extra bonus attack possible

2.5.2 Constructor

Name	Description
+ WildFighter(String type, int team)	- call super constructor with parameters - set name to “Wild” - set bonus stats

2.5.3 Methods

Name	Description
+ void setSpecialAbility()	does nothing. WildFighter have no special ability
+ int getSymbol()	return symbol depending on team

	and attacking type
+ void setBonusAttack()	- set bonus attack - set new attack
- void setBonusAttack()	set bonus attack with MIN_BONUS_ATTACK and MAX_EXTRA_BONUS_ATTACK

3. Package entity.base

3.1 abstract class Fighter implements Updatable

Fighter is a super class of Fighters in package entity. Fighter can do 1 action in a turn, Move or Attack, or Heal for HealerFighter. Fighters have base stats depending on attacking type, melee or range. But this class can not be initiated because every Fighter must have their special abilities or stats.

3.1.1 Fields

Name	Description
# String type	attacking type, melee or range
# String name	Fighter type by special ability or stats
# double attack	damage dealt per attack
# double defense	reduce damage income from an attack by defense percent Example: get attacked 20 point, if has defense 10, hit point will decreased by 18 point
# int attackRange	longest range that this Fighter to attack
# double hitPoint	current hit point

# double maxHitPoint	maximum hit point
# int totalMoves	longest distance that can move to per move
# int team	side, 1 or 2
# boolean isAlive	if hit point > 0, this is true, else false
# boolean isVisible	if is not alive, this is false
# boolean isReady	if Fighter ready to take action or not (still can take an action)
# Coordinate coordinate	where this fighter at
- final int BASE_MELEE_ATTACK	base attack stat of melees
- final int BASE_MELEE_DEFENSE	base defense stat of melees
- final int BASE_MELEE_MAXHITPOINT	base maxHitPoint stat of melees
- final int MELEE_ATTACK_RANGE	melee attackRange is 1
- final int BASE_RANGE_ATTACK	base attack of ranges
- final int BASE_RANGE_DEFENSE	base defense of ranges
- final int BASE_RANGE_MAXHITPOINT	base maxHitPoint of ranges
- final int RANGE_ATTACK_RANGE	range attackRange is 2
- final int BASE_TOTAL_MOVES	base total moves is 1

3.1.2 Constructor

Name	Description
+ Fighter(String type, int team)	- set fields from parameters - set coordinate to null - set base stats - set special ability

3.1.3 Methods

Name	Description
+ double attack(Fighter target)	- deal damage to target - update target - set ready to false - return damage done
+ void move(Coordinate targetCoordinate)	move to target Coordinate and set coordinate to match
- void setBaseStats()	set base stats depending on attacking type with constants, also set this Fighter alive, visible and ready
# abstract void setSpecialAbility()	set special ability for different Fighters
+ abstract int getSymbol()	get specific symbol to draw
# void setAttack(double attack)	set attack to parameter
# double getDefense()	return defense
# void setDefense(double defense)	set defense to parameter
+ double getHitPoint()	return hitPoint
+ void setHitPoint(double hitPoint)	set hitPoint to parameter if param < 0; set hitPoint to 0 if param > maxHitPoint; set hitPoint to maxHitPoint
+ String getType()	return type
+ void setType(String type)	set type to parameter

+ boolean isAlive()	return isAlive
- void setAlive(boolean isAlive)	set isAlive to parameter
- void setVisible(boolean isVisible)	set isVisible to parameter
+ int getAttackRange()	return attackRange
- void setAttackRange(int attackRange)	set attackRange to parameter
+ int getTotalMoves()	return totalMoves
# void setTotalMoves(int totalMoves)	set totalMoves to parameter
+ double getMaxHitPoint()	return maxHitPoint
# void setMaxHitPoint(double maxHitPoint)	set maxHitPoint to parameter
+ int getTeam()	return team
- void setTeam(int team)	set team to parameter
+ boolean isReady()	return isReady
+ void setReady(boolean isReady)	set isReady to parameter
+ Coordinate getCoordinate()	return coordinate
+ void setCoordinate(Coordinate coordinate)	set coordinate to parameter. and setFighter of the parameter to this Fighter
+ String getName()	return name
# void setName(String name)	set name to parameter
+ void update()	- check if hitPoint <= 0; set alive to false - check if not alive; set visible to false

3.2 interface HitPointRegenerable

Ability for ToughFighter and HealerFighter. Their hit point will be slightly increased every end of a turn.

3.2.1 Methods

Name	Description
+ void regenerateHitPoint()	increase Fighter's hitPoint for a specific amount every end of a turn

3.3 interface StatsIncreasable

This gives bonus stats for ToughFighter and WildFighter.

3.3.1 Methods

Name	Description
+ void setBonusStats()	increase base stats by specific Fighter's bonus stats

4. Package exception

4.1 class InvalidNameException extends Exception

4.1.1 Field

Name	Description
- String type	Description of type name error

4.1.2 Construtor

Name	Description
+ InvalidNameException(String	Set type of an error and print it out

type)	in console

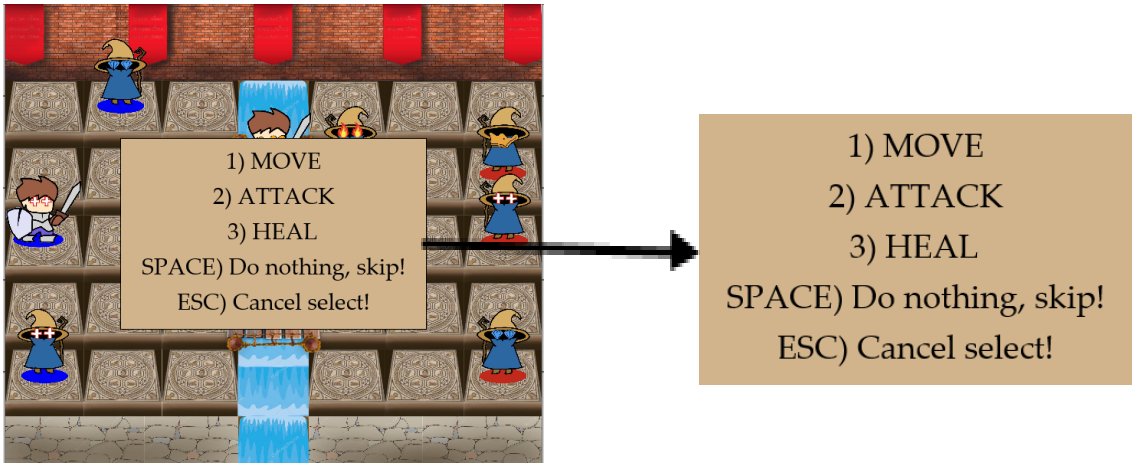
4.1.3 Method

Name	Description
+ String getType()	Get an error message type

5. Package gui

5.1 class **ActionPane** extends VBox

This class is the box that shows what action fighter can do in BoardPane



5.1.1 Field

Name	Description
- int addition	Addition number to be add in button control text
- Text moveText	Show the button to move
- Text attackText	Show the button to attack
- Text healText	Show the button to heal
- Text blackActionText	Show the button to skip
- Text cancelText	Show the button to cancel

5.1.2 Constructor

Name	Description
+ ActionPane(boolean isHealer, int moveCount, int attackCount, int healCount)	Constructor method. - sets max size to (400,250). - sets the alignment to center. - sets the background with Color.TAN - sets spacing to 15. - sets the border with Color.Black and its width to 1

	- initialize setup by calling setUp. - initialize text by calling setText. - adding the text by calling addText.

5.1.3 Method

Name	Description
- void setUp()	Check if the turn is player 1 - sets cancelText with text “ESC) Cancel select!” - sets the additional field to 0. Else if the turn is player 2 - sets cancelText with text “BACKSPACE) Cancel select!” - sets the additional field to 6.
- void setText()	- sets all text fields font with ("Palatino Linotype", FontWeight.SEMI_BOLD, 30). - sets moveText with “Integer.toString(1 + addition) + ") MOVE". - sets attackText with “Integer.toString(2 + addition) + ") ATTACK" - sets healText with “Integer.toString(3 + addition) + ") HEAL" - sets blankActionText with “SPACE) Do nothing, skip!”
- void addText(boolean isHealer, int moveCount, int attackCount, int healCount)	- Add moveText if moveCount > 1 - Add attackText if attackCount > 0 - Add healText if isHealer is true and healCount > 0 - Add blankActionText and cancelText.

5.2 class BoardPane extends Canvas implements Updatable

This class is the board where all fighters are standing at.

5.2.1 Field

Name	Description
- GraphicsContext gc	GraphicsContext for the board
- int[] pixel	The coordinate in pixel where the user are pointing
- char key	Key for an action
- ArrayList<Coordinate> coordinates	List of all coordinates that can do an action
- Coordinate beforeActionCoordinate;	Coordinate of its fighter
- Coordinate targetCoordinate	Coordinate of target fighter
- Coordinate actorCoordinate	Coordinate of actor fighter
- GameBoard gameBoard	GameBoard of the board
- ActionPane actionPane	ActionPane of fighter's action

5.2.2 Constructor

Name	Description
+ BoardPane()	Constructor method. - sets width to 700 - sets height to 600 - initialize the gc field by calling getGraphicsContext2D. - initialize gameBoard by getting from GameController.getGameBoard. - draw the board by calling draw method.

5.2.3 Method

Name	Description
- void draw()	Clear the board and draw board background.
+ void update()	Receive an action from users to do action if it is player 1 turn - Before selecting the fighter, stroke the box with Color.BLUE, LineWidth to 3 and use “W”,”A”,”S”,”D” to select the box of fighters to do an action. Use “F” key to select and showing ActionPane by add new ActionPane(actorCoordinate.getFighter() instanceof HealerFighter, gameBoard.getAllPossibleToMoveCoordinate(actorCoordinate).size(), gameBoard.getAllPossibleTargetsToAttack(actorCoordinate).size(), gameBoard.getAllPossibleAlliesToHeal(actorCoordinate).size()) - Selecting action to do that are showing in ActionPane

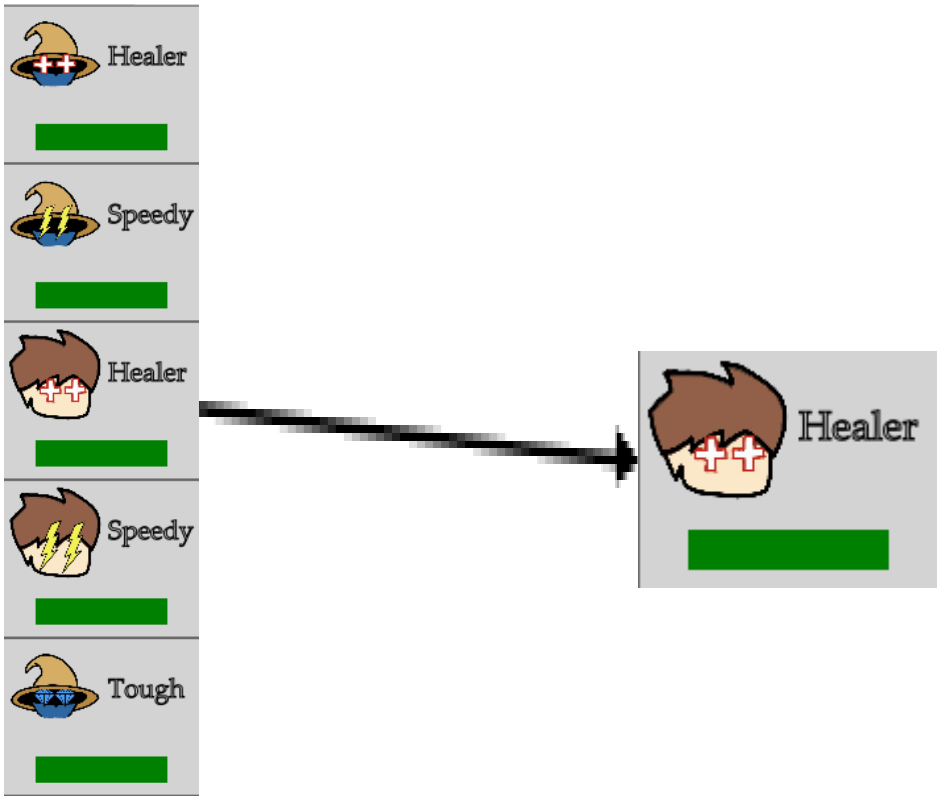
	- Selecting target fighter by using “W”,”A”,”S”,”D” key to select the box of target and “F” key to do an action to target else if it is player 2 turn - Before selecting the fighter, stroke the box with Color.BLUE, LineWidth to 3 and use “I”,”J”,”K”,”L” to select the box of fighters to do an action. Use “;” key to select and showing ActionPane by add new ActionPane(actorCoordinate.getFighter() instanceof HealerFighter, gameBoard.getAllPossibleToMoveCoordinate(actorCoordinate).size(), gameBoard.getAllPossibleTargetsToAttack(actorCoordinate).size(), gameBoard.getAllPossibleAlliesToHeal(actorCoordinate).size()) - Selecting action to do that are showing in ActionPane - Selecting target fighter by using “I”,”J”,”K”,”L” key to select the box of target and “;” key to do an action to target
- Coordinate getMovingToCoordinate(String direction)	using the class field actorCoordinate which in this method it is the moving Fighter - move to adjacent coordinate that actorCoordinate can move to, receiving direction as “left”, “top”, “right” or “bottom”
- Coordinate getTargetCoordinate(String direction, Coordinate currentCoordinate)	get targetCoordinate by considering direction and currentCoordinate - Coordinates that are straight to currentCoordinate will be selected

	first by method sortCoordinateByStraight() (below) - then consider by distance of the Coordinates and currentCoordinate by method sortByDistance() (below)
- void doMove(Coordinate targetCoordinate)	move the Fighter in actorCoordinate field to targetCoordinate
- void select(Coordinate currentCoordinate)	mark the currentCoordinate with strokeRect to tell that is selecting
- void sortCoordinateByStraight(Coordinate currentCoordinate)	sort coordinates field by straight between each Coordinate and currentCoordinate - straights to currentCoordinate come first and sorted by distance from method sortByDistance()
- static ArrayList<Coordinate> sortByDistance(ArrayList<Coordinate> coordinates, Coordinate currentCoordinate)	return sorted coordinates param by distance from currentCoordinate param, closest to the most far. - this method is used for selecting targets to take actions logic - it's hard to explain exactly what this method's objective is, but I will try to tell you that when Player uses WASD (for Player1 for example) to select boxes, you will get the closet box first.
+ void resetActorCoordinate()	this is called every end of a turn - assign actorCoordinate to null
+ void checkUnDoneMove()	check if there is an undone move (moved but not committed) when the turn time runs out - move it back to where it is before the move

	note move action needs commitment to done the move

5.3 class FighterBoxBattle extends FighterBox implements Updatable

This class is a box of fighter's head, name and hiit point bar that fills in PlayerPaneBattle. The hit point bar will decrease if the fighter gets damaged and increase when healed.



5.3.1 Construtor

Name	Description
+ FighterBoxBattle(Fighter fighter, int symbol)	Initialize super class - set width to 150 - set height to 120 - set Image by calling setImage() - draw components by calling

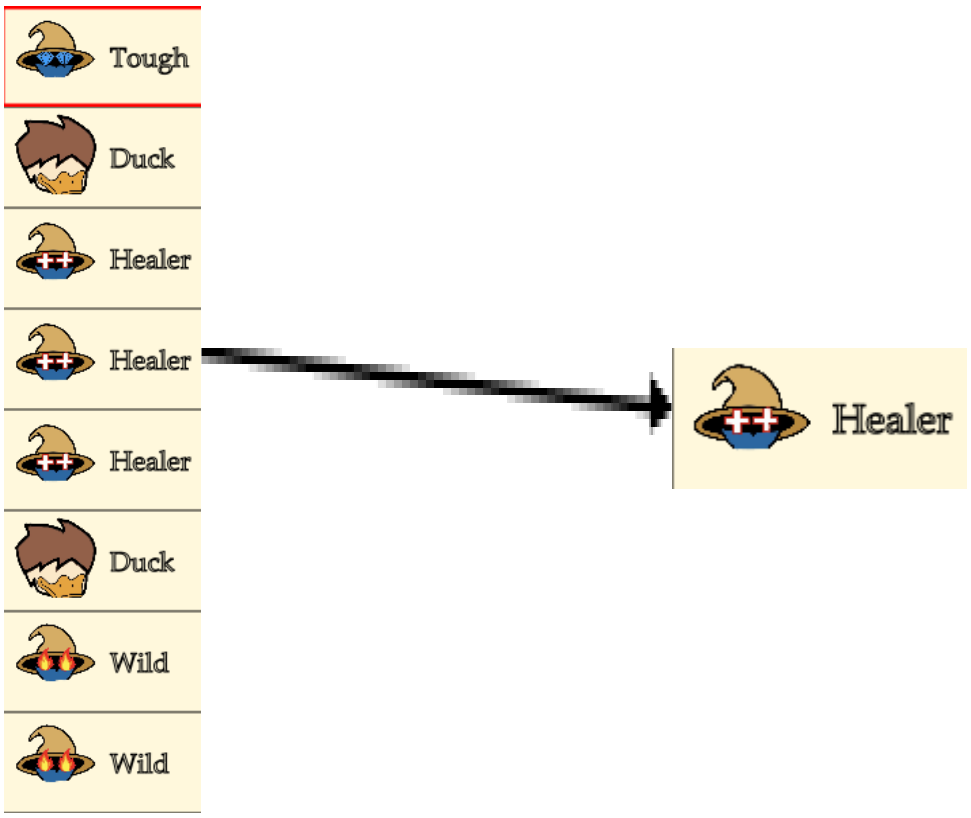
	draw()

5.3.2 Method

Name	Description
+ void draw()	Fill color with light gray Draw image and name Draw hit point bar
+ void setImage()	Calling RenderableHolder.getHeadImage(symbol)
+ void update()	Calling draw() And draw black bar for decreasing hit point

5.4 class FighterBoxPreBattle extends FighterBox implements Updatable

This class is a box of fighter's head and name which is fills in PlayerPanePreBattle to be select in pre battle phrase



5.4.1 Construtor

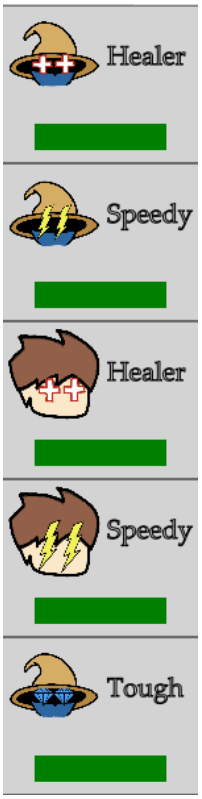
Name	Description
+ FighterBoxPreBattle(Fighter fighter, int symbol)	Initialize super class Set width to 150 Set height to 600/8 Calling setImage() and draw()

5.4.2 Method

Name	Description
+ void draw()	Fill background with cornsilk color Draw image and name Stroke box
+ void setImage()	Calling RenderableHolder.getHeadImage(symbol)
+ void update()	Stroke box with red color

5.5 class PlayerPaneBattle extends PlayerPane implements Updatable

This class is a status bar of the team that will automatic update when fighter got damaged or healed



5.5.1 Field

Name	Description
- ArrayList<Fighter> fighters	List of selected fighters

5.5.2 Construtor

Name	Description
+ PlayerPaneBattle(int player)	Initialize super class and fighters field

5.5.3 Method

Name	Description
+ void addFighters(Fighter fighter)	If fighters field size less than or equal to 5

	- Set fighter to the map in GameBoard - Add fighter to fighters - Add fighter box by calling addFighterBox()
+ void addFighterBox(Fighter fighter)	Add children with new FighterBoxBattle
+ void update()	Looping to update all boxes
+ void fillUp(PlayerPanePreBattle playerPreBattle)	Add fighter until fighter size equals to 5 by calling playerPreBattle.getFightersLeft()
+ ArrayList<Fighter> getFighters()	Get the list of fighters

5.6 class PlayerPanePreBattle extends PlayerPane implements Updatable

This class is a bar that shows random fighters.Player will choose 5 fighters from 8 fighters here. If the player didn't select the fighter in time, it will automatically select from the top of the pane to fill up the board.

5.6.1 Field

Name	Description
- Fighter[] fighters	List of random fighters
- FighterBoxPreBattle unselectedPreFighter	Unselected fighter box
- FighterBoxPreBattle selectedPreFighter	Selected fighter box

5.6.2 Construtor

Name	Description
+ PlayerPanePreBattle(int player)	- Initialize super class - Initialize fighters by calling LogicUtility.getRandomFighters(player) - create a box for all fighters and add it by calling addFighterBox

5.6.3 Method

Name	Description
+ void addFighterBox(Fighter fighter)	- Add children with new box of fighter
+ void update()	Check the player side and player's fighter box size to be more than 3 - Use "W" for player 1 and "I" for player 2 to move cursor up - Use "S" for player 1 and use "K" for player 2 to move cursor down - stroke the selected box - Use "F" for player 1 and ";" for player 2 to select the fighter and add it to the PaneBattle

+ Fighter[] getFightersLeft()	Get the fighter from the top of the pane until PaneBattle of each player has 5 fighters

5.7 class SimulationManager

5.7.1 Field

Name	Description
- static PlayerPaneBattle P1PaneBattle	Player 1 Pane battle
- static PlayerPaneBattle P2PaneBattle	Player 2 Pane battle
- static PlayerPanePreBattle P1PanePreBattle	Player 1 Pane pre battle
- static PlayerPanePreBattle P2PanePreBattle	Player 2 Pane pre battle
- static BoardPane board	Board of the game

5.7.2 Method

Name	Description
+ static void initializeAllPane()	Initialize all fields
+ static void updatePaneBattle()	Update board and both pane battle
+ static void updatePanePreBattle()	Update bothe pre pane battle
+ static PlayerPaneBattle getP1PaneBattle()	Get the player 1 pane battle
+ static PlayerPaneBattle getP2PaneBattle()	Get the player 2 pane battle
+ static PlayerPanePreBattle	Update selected player 1 fighter

getP1PanePreBattle()	box and return player 1 pane pre battle
+ static PlayerPanePreBattle getP2PanePreBattle()	Update selected player 2 fighter box and return player 2 pane pre battle
+ static BoardPane getBoard()	Return the board
+ static void fillUpPaneBattle()	Fill up both pane battles and set fighters of each player

6. Package gui.base

6.1 class **ActionButton** extends Button

6.1.1 Construtor

Name	Description
+ ActionButton(String name)	Initialize super class - set background with BackgroundFill(Color.LIGHTGREY, CornerRadii.EMPTY, Insets.EMPTY)) - set preferred size to (300, 75) - set alignment to center - set border with BorderStroke(Color.BLACK, BorderStrokeStyle.SOLID, null, new BorderWidths(1)) - set font with “Palatino Linotype” and font size 30 - set up action by calling setUpAction

6.1.2 Method

Name	Description
- void setUpAction()	- Set on mouse entered change background with white color and

	set drop shadow - Set on mouse exited change background with light gray color and set effect to be null

6.2 abstract class **FighterBox** extends Canvas

6.2.1 Field

Name	Description
# int symbol	Symbol of the fighter
# GraphicsContext gc	GraphicsContext of the box
# Image image	Image of fighter's head
# Fighter fighter	Fighter of the box

6.2.2 Construtor

Name	Description
+ FighterBox(Fighter fighter, int symbol)	Initialize fighter, symbol, and gc fields

6.2.3 Method

Name	Description
+ *void draw()*	Draw the box
+ *void setImage()*	Set the image of fighter's head
+ Image getImage()	Get the image
+ Fighter getFighter()	Get the fighter

6.3 abstract class **PlayerPane** extends VBox

6.3.1 Field

Name	Description
# int player	The player side (1 or 2)

6.3.2 Construtor

Name	Description
+ PlayerPane(int player)	Initialize player field - set background with BackgroundFill(Color.GRAY, CornerRadii.EMPTY, Insets.EMPTY) - set preferred size to (150,600)

6.3.3 Method

Name	Description
+ abstract void addFighterBox(Fighter fighter)	Add the fighter box to the pane

7. Package input

7.1 class InputUtility

This class is used to collect input action from users

7.1.1 Field

Name	Description
- static ArrayList<KeyCode> keyPressed	List of key pressed from user

7.1.2 Method

Name	Description
+ static boolean	Check if the key got pressed or

getKeyPressed(KeyCode keycode)	not
+ static void setKeyPressed(KeyCode keyCode, boolean pressed)	Adding keyCode in to the keyPressed ArrayList if it is not already pressed
+ static void removeKeyPressed()	Removing all the keyCode in the keypressed ArrayList.

8. Package logic

8.1 class Coordinate implements Updatable, Comparable<Coordinate>

Coordinate represent the position of a box on the board.

8.1.1 Fields

Name	Description
- int i	row index
- int j	column index
- boolean isEmpty	is this Coordinate empty
- Fighter myFighter	Fighter in this Coordinate

8.1.2 Constructor

Name	Description
+ Coordinate(int i, int j)	- set fields to parameters - set myFighter to null. Coordinate is initiated empty

8.1.3 Methods

Name	Description

+ int getI()	return i
- void setI(int i)	set i to parameter
+ int getJ()	return j
- void setJ(int j)	set j to parameter
+ Fighter getFighter()	return myFighter
+ void setFighter(Fighter fighter)	- set myFighter to parameter - set empty to false if myFighter is null, else true
+ int getSymbol()	- if myFighter is null and this is empty; return GROUND symbol - if myFighter is null but this is not empty; return RIVER symbol - else return myFighter's symbol
+ boolean isEmpty()	return isEmpty
- void setEmpty(boolean isEmpty)	set isEmpty to parameter, but if this Coordinate is river's position, this Coordinate will not be empty; isEmpty is always false for river
+ void update()	if myFighter is null; set Empty to true
+ int[] coordinate2Pixel()	return array of 2 Integers which is real position on the screen by converting i and j to pixel position
+ String toString()	return String of (i,j)
+ int compareTo(Coordinate other)	return compare of this Coordinate and other Coordinate's Strings

8.2 class EffectUtility

This class is to help choose the right effect symbols to display, or play sound effects from actions and actors.

8.2.1 Methods

Name	Description
+ static int getEffectSymbol(char key, Coordinate actorCoordinate, double damageDone)	- key refers to action, attack or heal - actorCoordinate is Coordinate of actor - damageDone is true damage dealt, if it's heal action this receive 0 - check what effect will be displayed by action, actor's team and type, and damageDone. If it's attack action but damageDone is zero it means DuckFighter dodged the attack
+ static void playSoundEffect(char key, Coordinate actorCoordinate, double damageDone)	- works just like getEffectSymbol() but this will play sound of an action

8.3 class GameBoard implements Updatable

GameBoard holds all Coordinates on the board and all Fighters. Also control the actions of Fighters.

8.3.1 Fields

Name	Description
- Coordinate[][] board	all Coordinates on the board
+ int[][] map	symbol for each Coordinate
- final int ROWS	number of rows
- final int COLS	number of columns
+ ArrayList<Fighter>	Player1's Fighters alive

Player1Fighters	
+ ArrayList<Fighter> Player2Fighters	Player2’s Fighters alive

8.3.2 Constructor

Name	Description
+ GameBoard()	- init board and map fields by ROW and COLS - call setDefault()

8.3.3 Methods

Name	Description
+ void setPlayerFighters(ArrayList<Fighter> fighters1, ArrayList<Fighter> fighters2)	set fighters received from selected fighters - fighters1 and fighters2 are Player1’s fighters and Player2’s fighters respectively - add fighters to the board same order as parameters
+ void setDefault()	is called when new round started - initiate new Player1Fighters and Player2Fighters - call setBoardandMap()
- void setBoardAndMap()	initiate Coordinates and set to board and map fields
+ ArrayList<Coordinate> getAllReadyFightersCoordinate(int team)	return all Fighters that is ready to take action by parameter team
+ ArrayList<Coordinate> getAllPossibleToMoveCoordinate(Coordinate currentCoordinate)	return all Coordinates that Fighter in currentCoordinate can move to
+ ArrayList<Coordinate> getAllPossibleTargetsToAttack(Coordinate attackerCoordinate)	return all Fighters’ Coordinate that Fighter in attackerCoordinate can attack

+ ArrayList<Coordinate> getAllPossibleAlliesToHeal(Coordinate healerCoordinate)	return all Fighters' Coordinate that Fighter in healerCoordinate can heal
+ void takeMove(Coordinate actorCoordinate, Coordinate targetCoordinate)	move Fighter in actorCoordinate to targetCoordinate - update actorCoordinate and targetCoordinate
+ double takeAttack(Coordinate attackerCoordinate, Coordinate targetCoordinate)	make Fighter in attackerCoordinate attack Fighter in targetCoordinate - update attackerCoordinate and targetCoordinate - return absolute damage done
+ double takeHeal(Coordinate healerCoordinate, Coordinate targetCoordinate)	make Fighter in healerCoordinate heal Fighter in targetCoordinate - update healerCoordinate and targetCoordinate - return total healing point done
- void addFighter(Fighter fighter, Coordinate coordinate, int team)	- add param fighter to param coordinate to list of Fighters by team - set the coordinate to the fighter and the fighter to the coordinate - update the coordinate
- void removeFighter(Coordinate coordinate)	- remove Fighter in param coordinate from the list - set the Fighter's coordinate to null and the coordinate's Fighter to null - update the coordinate
- ArrayList<Coordinate> getAllPossibleToMoveDistance1(Coordinate currentCoordinate)	return all possible Coordinates that Fighter in currentCoordinate can move to. Looking only adjacent Coordinates to currentCoordinate (1 box far)
- ArrayList<Fighter> getFightersByTeam(int team)	return Fighters by the given team

+ int[][] getMap()	return map
+ void printMap()	print map (symbols)
+ Coordinate[][] getBoard()	return board
- void updateMap(Coordinate coordinate)	update symbol of the given coordinate
- void update(Coordinate coordinate)	check if Fighter in coordinate alive, if there is Fighter; if not, remove the Fighter by using removeFighter(Coordinate coordinate)
- void update(Coordinate coordinate1, Coordinate coordinate2)	just for updating two Coordinates by calling once - call update(Coordinate coordinate) for both params
+ void update()	called every end of a turn - regenerate hit point for those have the ability
+ void resetReady()	set all the Fighters ready

8.4 class GameConstant

This class holds game constants.

8.4.1 Fields

Name	Description
+ static final int ORIGIN_X	the game's board starting point on the screen
+ static final int ORIGIN_Y	the game's board starting point on the screen
+ static final int BOX_WIDTH	width of a box in the board on the screen

+ static final int BOX_HEIGHT	height of a box in the board on the screen
+ static final int N_ROWS	number of rows of the board
+ static final int N_COLS	number of columns of the board
+ static final int PRE_BATTLE_PHASE_TIME	total time given to choose Fighters in pre battle phase
+ static final int BATTLE_PHASE_TIME_PER_TURN	time given to decide and take actions of Fighters in a turn
+ static final int MAX_TURN_PER_PLAYER	maximum turns per round per player
+ static final int MAX_ROUND	maximum rounds per game
+ static final String MELEE_TYPE_STRING	String that refers to Fighter's type "melee"
+ static final String RANGE_TYPE_STRING	String that refers to Fighter's type "range"
+ static final int TEAM_1	int that refers to team 1
+ static final int TEAM_2	int that refers to team 2
+ static final String DUCK_NAME	name of DuckFighter "Duck"
+ static final String HEALER_NAME	name of HealerFighter "Healer"
+ static final String SPEEDY_NAME	name of SpeedyFighter "Speedy"
+ static final String TOUGH_NAME	name of ToughFighter "Tough"
+ static final String WILD_NAME	name of WildFighter "Wild"
+ static final char MOVE_KEY	char key refers to move action
+ static final char ATTACK_KEY	char key refers to attack action
+ static final char HEAL_KEY	char key refers to heal action

8.5 class GameController

This class controls game processes

8.5.1 Fields

Name	Description
- static GameBoard gameBoard	GameBoard of the game
- static int P1Score	Player 1’s scores obtained
- static int P2Score	Player 2’s scores obtained
- static int roundCount	current round’s number
- static int turnCount	current turn’s number
- static boolean isRoundDone	is current round done managing data
- static boolean isRoundOver	is current round got the winner, can be set from 2 situations 1) turn count exceed maximum turns or one Player is out of Fighters
- static boolean isGame	is the game over
- static boolean isWin	is the game got a winner meaning the game not tied
- static boolean isP1	is it Player1’s turn, otherwise it’s Player2’s turn
- static boolean isEndPreBattle	is the pre battle phase ended
- static boolean isTurnDone	is the turn over, meaning that the time is up or each of all Fighters are already done an action
- static boolean isSelected	is there a selected Fighter to take action
- static boolean isChose	is the selected Fighter is already chose an action
- static String P1Name	Player 1’s name

- static String P2Name	Player 2’s name
- static String winner	winner’s name

8.5.2 Methods

Name	Description
+ static void setDefault(String P1Name, String P2Name)	start the game - initiate new GameBoard and set to the field - set P1Score and P2Score to 0 - set roundCount and roundCount to 1 - set player names to the params - set isP1 to true, the game start with Player1 - set isGame, isSelected and isChose to false
+ static void update()	update every end of a turn First set some trackers and reset some fields - add turn count by 1 - set isSelected and isChose to false - set isTurnDone to true - change side playing by setP1 to the opposite of current side - set all Fighters ready - call update of gameBoard (regenerate hit point to those have the ability to) Second check if the round over - if turn count exceed maximum turns; setRoundOver to true - if is the round over; start new round, add round count, set some defaults, setRoundDone to true and setRoundOver back to false Lastly check if the game is over

	- if rounds exceed maximum rounds or if one Player have score of 2; the game ends and get the winner's name
+ static boolean checkRoundOver()	if one Player is out of Fighters; add the winner of the round a score and return true, otherwise return false
- static void checkWinnerAtTheEnd()	check winner at the end of a round. There are 2 steps 1) check number of Fighters left 2) if the numbers are the same then use percent of sum of hit point remains by sum of max hit point of each Players' Fighters
- static void checkWinner()	at the end of the game - check which Player got more score - set the winner if the both Players do not have the same score
- static void addRoundCount()	add roundCount by 1
- static void addTurnCount()	add turnCount by 1
+ getters/setters for each fields	

8.6 class LogicUtility

This class has utility methods including calculations, checking possibilities of actions and more.

8.6.1 Methods

Name	Description
+ static int calculateDisTance(Coordinate c1, Coordinate c2)	return distance between c1 and c2 - distance is how many boxes are far from c1 and c2. For example 1) if c1 is at (1,2) and c2 is at

	(3,2), distance is 2 2) if c1 is at (1,0) and c2 is at (2,3), distance is 4
+ static double calculatePercentSumOfHitPointRemain(ArrayList<Fighter> fighters)	return percent of (sum of hit point remain of every fighters) / (sum of max hit point of every fighters)
+ static boolean isPossibleToMoveTo(Coordinate targetCoordinate)	check if targetCoordinate can be moved to by a Fighter - consider emptiness of the targetCoordinate; if it's empty return true, otherwise return false
+ static boolean isPossibleToAttack(Fighter attacker, Fighter target)	check if attacker can attack target - consider attacker's attack range and distance between them; if target's in the range of attacker return true, otherwise return false
+ static boolean isPossibleToGetHealed(Fighter target)	check if target can get healed - consider target's hitpoint is at max hit point (full hit point); if it's at max hit point return false, otherwise return true
+ static Fighter getRandomFighter(int team)	generate random Fighter from 10 different types by team param and return it **note** that 10 different types are 5 special ability or stats and 2 attacking type

8.7 class Sprites

This class holds constant symbols of things to draw on the board , including Ground, River, Fighters, and action effects.

8.7.1 Fields

Name	Description

+ static final int P1_DUCKMELEE	1
+ static final int P1_HEALERMELEE	2
+ static final int P1_SPEEDYMELEE	3
+ static final int P1_TOUGHMELEE	4
+ static final int P1_WILDMELEE	5
+ static final int P1_DUCKRANGE	6
+ static final int P1_HEALERRANGE	7
+ static final int P1_SPEEDYRANGE	8
+ static final int P1_TOUGHRANGE	9
+ static final int P1_WILDRANGE	10
+ static final int P2_DUCKMELEE	11
+ static final int P2_HEALERMELEE	12
+ static final int P2_SPEEDYMELEE	13
+ static final int P2_TOUGHMELEE	14
+ static final int P2_WILDMELEE	15
+ static final int P2_WILDMELEE	16
+ static final int P2_DUCKRANGE	17
+ static final int P2_HEALERRANGE	18
+ static final int P2_SPEEDYRANGE	19

+ static final int P2_TOUGHRANGE	20
+ static final int RIVER	21
+ static final int GROUND	22
+ static final int P1_MELEEATTACK	23
+ static final int P2_MELEEATTACK	24
+ static final int RANGEATTACK	25
+ static final int HEAL	26
+ static final int P1_DUCKED	27
+ static final int P2_DUCKED	28

8.8 interface Updatable

This is for many classes that can be updated, which are not constant or static classes.

8.8.1 Methods

Name	Description
+ void update()	update

9. Package screen

9.1 class EndScreen

9.1.1 Field

Name	Description
- Stage primaryStage	Stage of this game
- Pane root	Root of this screen

- VBox textGroup	Text box that will show in the screen
- HBox buttonGroup	Group of button that groups with backButton and quitButton
- AnimationTimer endScreenSong	The timer that will play the sound of its screen
- Text congratulationText	Text fills with“CONGRATULATION”
- Text playerName	Text fills with the winner name
- Text tieText	Text fills with “TIE GAME”
- Button backButton	Button that will link back to main game
- Button quitButton	Button that will close the game

9.1.2 Constructor

Name	Description
+ EndScreen(Stage primaryStage)	Constructor method. - initialize primaryStage field. - initialize root field. - set root field max size to (1000,800) - calling setUpSongAndTextGroup and setUpButtonGroup - initialize scene and setScene to primaryStage

9.1.3 Method

Name	Description
- void setUpSongAndTextGroup()	Check if GameController,isWin is

	true 1. Play win song 2. Set background with end background 3. Initialize textGroup with 60 spacing 4. Set textGroup padding with Insets(0, 0, 400, 400) 5. Set textGroupAlignment to center 6. Set layoutX to 100 7. Set layoutY to 170 8. Initialize and add congratulationText with “Congratulation” and playerNameText with GameController.getWinner to textGroup 9. Set congratulationText with (Font.font("Palatino Linotype", FontWeight.SEMI_BOLD, 40)) and set playerNameText with Font.font("Palatino Linotype", FontWeight.SEMI_BOLD, 65)). 10. Set all Text to white color. 11. Set title to “Congratulation!!” Else 1. Play Tie song 2. Set background with tie background 3. Initialize textGroup with 30 spacing 4. Set textGroup padding with Insets(0, 0, 400, 400) 5. Set textGroupAlignment to center 6. Set layoutX to -90

	7. Set layoutY to 80 8. Initialize and add tieText with “TIE GAME” to textGroup 9. Set tieText with Font.font("Palatino Linotype", FontWeight.SEMI_BOLD, 80) 10. Set tieText to white color 11. Set title to “TIE GAME”
- void setUpButtonGroup()	- Initialize buttonGroup with spacing to 20 - Set padding with Insets(100) - Set layoutX to 370 - Set layoutY to 550 - Initialize backButton and quitButton with “Main Menu” and “”QUIT”, respectively - Set both buttons preferred size to (200,50) - Set both buttons font with Font.font("Palatino Linotype", FontWeight.SEMI_BOLD, 20) - Add backButton onMouseClicked to close song and start new main screen - Add quitButton onMouseClicked to close the game - Add all buttons to buttonGroup

9.2 class GameScreen

9.2.1 Field

Name	Description
- String P1Name	Player 1’s name
- String P2Name	Player 2’s name
- Stage primaryStage	Stage of the application
- Scene scene	Scene of the application
- BorderPane root	Root of the scene
- Canvas gameCanvas	Canvas of game board
- Canvas statusCanvas	Canvas of status bar
- HBox statusPane	Status bar
- StackPane namePane	Name bar
- PlayerPane P1Pane	Player 1 pane
- PlayerPane P2Pane	Player 2 pane
- GameBoard gameBoard	Game board
- AnimationTimer gameScreenSong	Timer of game song
+ static GraphicsContext gameGC	GraphicsContext of game board
+ static GraphicsContext statusGC	GraphicsContext of status bar
+ static int gameTime	Amount of time left
+ static int positionToSelectP1	Cursor of player 1
+ static int positionToSelectP2	Cursor of player 2
+ static int timeToDrawAnimation	Amount of time to draw
+ static int[] selectedPixel	Selected box to draw
+ static boolean P1	Side of player
+ static String	Type of selected fighter

selectedFighterType	
+ static int effectSymbol	Effect symbol
+ static StackPane board	Board of the game

9.2.2 Construtor

Name	Description
+ GameScreen(Stage primaryStage, String P1Name, String P2Name)	- Initialize stage and both player name fields - Initialize game board by calling GameController.getGameBoard() - Initialize board field - start the gameScreenSong - initialize game by calling initializeGame()

9.2.3 Method

Name	Description
- void paintGameScreenComponent()	Draw all fighters on the board
- void drawAnimation()	If the timeToDrawAnimation > 0, draw effect at selectedPixel and then decrease timeToDrawAnimation by 1 else do nothing
- void drawNamePane()	Initialize namePane - set namePane alignment to center - set naemPane preferred size to (1000,100) - set namePane background with gameNameBar_bg_Image - add gameName Text “BATT:E BOARD” to its children Set gameName Text with

	Font.font("Times New Roman", FontWeight.BOLD, 36) and set stroke with silver color
- void drawStatusPane()	Initialize statusPane - set statusPane max size to (1000,100) - initialize P1Tag, P2Tag - set P1Tag and P2Tag border with BorderStroke(Color.SILVER, BorderStrokeStyle.SOLID, null, new BorderWidths(1) - set P1Tag and P2Tag set preferred size to (100,100) - set P1Tag background with blue color and set P2Tag background with red color - create P1NameTag with “Player 1:\n” + P1Name and create P2NameTag with “Player2\n” + P2Name - add P1NameTag to P1Tag and P2NameTag to P2Tag - set both NameTag with Font.font("Times New Roman", FontWeight.BOLD, 20) and white color - initialize statusCanvas and statusGC - add P1Tag,statusCanvas,P2tag to statusPane
- void addListener(Scene scene)	Receive input from user
- void setScene()	Initialize root field - setTop with statusPane - setLeft with P1Pane - setRight with P2Pane - setCenter with board - setBottom with NamePane Initialize scene Calling addListener setTitle”Battle Board”

	setScene
- void drawTimeAndRound()	Draw Player 1 win ,Round - Turn and time, Player 2 win When time is less than 10 change from black to red
- void initializeGame()	- Initialize all pane - set player pane with pre battle pane - start time countdown - when time is up or end pre select fighter initialize battle - start animationTimer to update pane pre battle
- void initializeBattle()	- set Player pane with battle pane - start timePerTurn countdown stop if one of the team has no fighter left or time is up - start animationTimer to run each turn in the round if round is over calling initialize game

9.3 class HowToPlay extends VBox

9.3.1 Field

Name	Description
- Text title	“HOW TO PLAY”
- GridPane description	Description of the game

9.3.2 Construtor

Name	Description
+ HowToPlay()	Constructor method. - set alignment to center - set spacing to 50

	- set max size to (500,750) - set background with howToPlay_bg_Image - initialize description and title - create scrollPane and set description - set scrollPane max size to (450,450) - create back button with ActionButton and set on mouse click to close the pane - add title,scrollPane, and back button to its children

9.4 class **MenuBar** extends VBox

9.4.1 Field

Name	Description
+ Button startButton	Action button with “START GAME”
+ Button howToPlayButton	Action button with “HOW TO PLAY”
+ Button quitButton	Action button with “QUIT”

9.4.2 Construtor

Name	Description
+ MenuBar()	- set alignment to center - set preferred height to 300 - set preferred width to 180 - set spacing to 30 - set padding with Insets(200, 50, 200, 50) - initialize all buttons and add it

9.5 class **PlayerNameBar** extends VBox

9.5.1 Field

Name	Description
- Stage primaryStage	Stage of the application
- String Player1Name	Player 1’s name
- String Player2Name	Player 2’s name
- TextField Player1NameField	Name field of player 1
- TextField Player2NameField	Name field of player 2
- HBox nameConfirm1	Name group box of player 1
- HBox nameConfirm2	Name group box of player 2
- HBox buttonBox	Button box
- Button confirmButton1	Confirm button of player 1
- Button confirmButton2	Confirm button of player 2
- Button editButton1	Edit button of player 1
- Button editButton2	Edit button of player 2
- Button startButton	Start button
- Button backButton	Back button
- boolean isCheckP1	Is player 1’s name confirm
- boolean isCheckP2	Is player 2’s name confirm

9.5.2 Construtor

Name	Description
+ PlayerNameBar(Stage primaryStage)	Initialize stage - set border with BorderStroke(Color.BLACK, BorderStrokeStyle.SOLID, null, new BorderWidths(1)) - set alignment to center - set max size to (720,300)

	- set padding with Insets(100, 50, 100, 50) - set spacing to 25 - set background with BackgroundFill(Color.TAN, CornerRadii.EMPTY, Insets.EMPTY) - set up button and name confirm - create description text - create stack pane to group description text set its alignment to center - create player 1 text and player 2 text - add children with p1, nameConfirm1, p2, nameConfirm2, buttonBox, groupText, respectively

9.5.3 Method

Name	Description
- void setUpTextFields()	Initialize all text fields - set preferred size to(200,50) - set font with "Palatino Linotype", FontWeight.SEMI_BOLD, 20
- void setUpNameConfirm()	Calling setUpTextFields - initialize all nameConfirm with spacing 30 - add nameField and confirmButton to its nameConfirm
- void confirmName(String name) throws InvalidNameException	Checking valid names. The name should be as following: 1) The name shouldn’t be blank. 2) The name should contain only English alphabet or number or space 3) The name should not exceed 6 characters

	4) The name should not be the same as another player.
- void setUpButtons()	- set up on mouse click action of each button Confirm button - play click sound and check the name if it's not check set player name and set isCheck is true change from confirm button to edit button and also add check image Edit button - play click sound and remove text field, check image and also change back to confirm button Start button can be click only when isCheck of both player are true if it is true go to game screen Back button close the name pane and back to main screen
- ImageView imageViewCheck()	Set up check image and return its image

9.6 class StartScreen

9.6.1 Field

Name	Description
- Stage primaryStage	Stage of the application
- Canvas canvas	Canvas of the start screen
- GraphicsContext gc	GraphicsContext of the start screen
- MenuBar menu	Menu that contains button
- HowToPlay howToPlay	How to play pane
+ static StackPane root	Root of the screen
+ static AnimationTimer	Timer for the song

startScreenSong	

9.6.2 Construtor

Name	Description
+ StartScreen(Stage primaryStage)	Initialize all fields and calling setUp method

9.6.3 Method

Name	Description
+ void setBackground(GraphicsContext gc)	Draw background image and draw title name and background
+ void drawScene(GraphicsContext gc)	Add scene and set scene - start the song - set title “Battle Board”
+ void setMenuAction()	Add on mouse click action of each button Start button click to open the name pane bar How to play button click to open the how to play pane Quit button to close the application
+ static void removeChildFromRoot()	Remove the last children of the root
+ void setUp()	Calling setMenuAction, setBackground and drawScene

10. Package sharedObject

10.1 class RenderableHolder

10.1.1 Field

Name	Description
+ static IMAGES	All images used in this application
+ static AUDIO	All audio used in this application

10.1.2 Method

Name	Description
+ static Image getHeadImage(int symbol)	Return head image of the fighter by symbol
+ static Image getFullBodyImage(int symbol)	Return full body image of the fighter by symbol
+ static Image getEffectImage(int symbol)	Return effect of the fighter action by symbol
+ static void loadResource()	Loading all image with Classloader