RAD

1 Introduction

Last summer the whole world saw the enormous success of *Pokémon GO* swipe across the world. The genre of this game, and its "predecessor" *Ingress*, is not a very discovered genre and we believe that we could bring a new touch to it. The idea is a location-based game, like the two games from Niantic, but focused more on interaction between players instead of interactions with selected locations in the real world. The way we thought we could approach this is in a combat-focused manor where you can attack players that you find in real life, via the game of course, but you have to be prepared for counterattacks at the same time. Things that could add some variation to the game are for example different classes, weapons and armour which can be acquired and impact how much damage you can take and give.

Killing other players award you with an increase in rank and a chance to find loot, while they lose both rank and gear. The idea is that the killed player drops a bag of loot which is available to pick up by nearby players. This might cause minor skirmishes between players hungry for loot, so one might have to take on a fight in pursuit of loot.

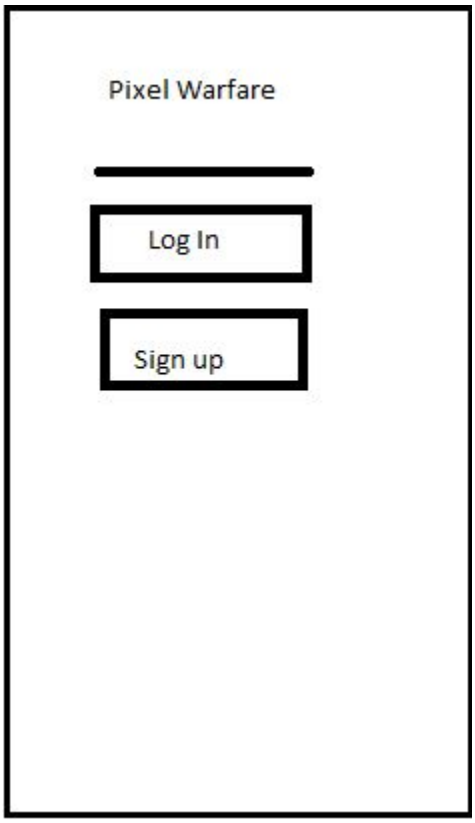
1.2 Definitions, acronyms and abbreviations

- HP: Health Points, a percentage often shown as a health bar which decreases as you take damage.
- Loot: Common game term for items dropped by a defeated foe, npc or fellow player.
- Online mode: A mode in which you can interact with and see other players.
- Offline mode: A mode in which all your interactive abilities are disabled and where you can't see or be seen by other players.
- Exp: Short for Experience Points which is used to gain ranks.

2 Requirements

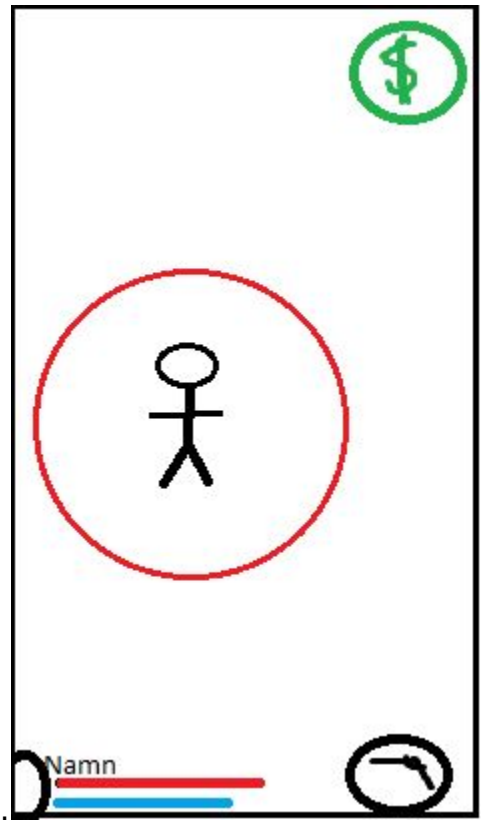
2.1 User interface

The interface will be described here along with some sketches of the different views. When opening the application the user is first met by a basic Login-view which is used to log in to the game.



When logged in the user sees a map, commonly styled like those offered by Google's map services for Android phones.. An avatar representing the user will be displayed on the map, surrounded by a red semi-transparent circular area which represents the shooting range of the currently equipped weapon . Whenever a user goes online, other online users will appear as they enter the view range, and users will be able to shoot other players once they come within the weapon range.

The interface also displays the user's name, avatar-picture, rank, current HP and current Exp in the lower left corner. On the other side the currently equipped weapon is displayed, inside of a circle. When clicked the interface displays a list of all the weapons that the user has in his inventory, and provides the options to equip any other weapon that the user possess



In the upper right corner of the interface there is a button which brings the player to the shop. The shop consists of an interface that is easy to understand and shows which weapons and armors that are avaliable for purchase. For each and everyone of these items there is a buy-button which is used for purchasing the item. There is also a sell-function for any items that the player owns.

2.2 Functional requirements

The user should be able to:

1. Move his/hers avatar, by actually walking around
2. See other players avatars based on proximity
3. Deal damage to other players avatars , with equipped weapon
4. Kill other players avatars
5. Take damage from other players avatars
6. Get his/her avatar killed
7. Gain rank
8. Lose rank
9. Maintain an inventory
    a. Collect items
        i. Buy from shop
        ii. Loot killed enemies
    b. Lose items when killed
10. Equip/Unequip items
11. Toggle between online and offline mode

2.3 Non-functional requirements

2.3.1 Usability

The application and its mechanics should be easily understood by a large margin of all users after reading description at the store and after playing the game for 10 minutes. Whereas the mechanics are simple they should allow for more complex strategies to be learnt over time.

As the game is meant to be played by actual movement and by interacting with other real people, it won't be the quick in-and-out type of experience that can be used at any microbreak. The game will be experienced optimally outside with at least 2 hectares available for movement.

2.3.2 Reliability

Reliability is not considered for this game as it's meant to used at leisure.

2.3.3 Performance

The application features competition between human players where reflexes and timing can be crucial, therefore good responsiveness is important so that it feels fair. As the graphics will be simple the likely bottleneck will be networking. Each frame should take at least 33 ms to render. And each involved client should poll the server with 33 ms intervals when players can see each other.

2.3.4 Supportability

Not applicable.

2.3.5 Implementation

The application will be written in Java and designed for Android. However, only the clientside is written for Android, which means that the serverside (with exceptions to the GoogleMaps-related packages, which are Android-specific) is able to be used in conjunction with another client on other platforms. This is because the application is written in such a way that the clientside and the serverside can be easily separated.

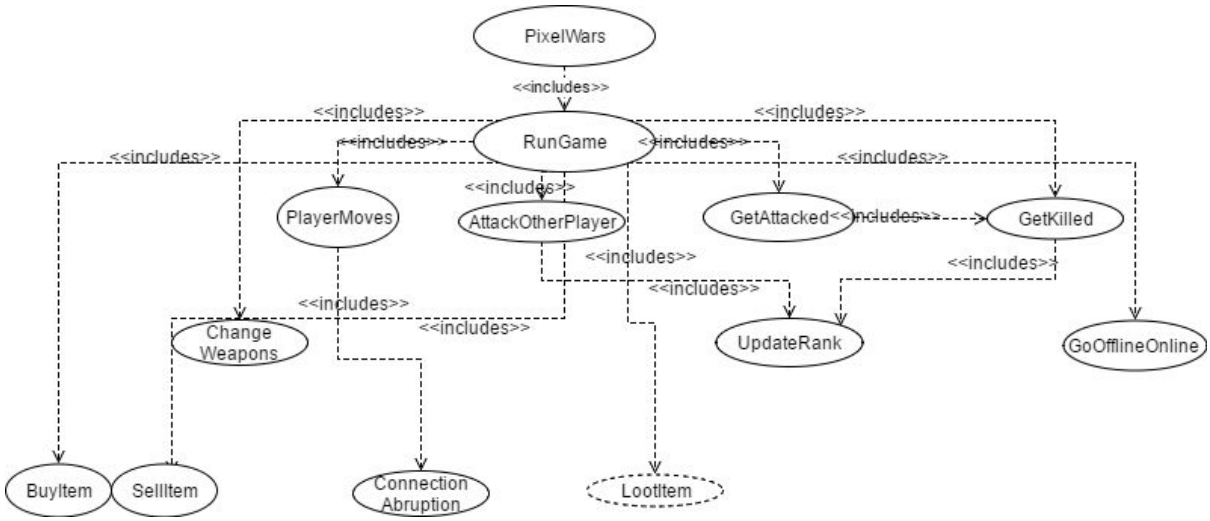
2.3.6 Packaging

The application will be developed for Android and as such packaged as an .apk file.
The Server program will be exported as a .jar file and can be on a hosting machine with Java Virtual Machine.

2.3.7 Legal

Google, Inc. allows developers to use the Google Maps-functionality for free, with no ads required.

3 Use cases



3.1 Use case listing

Use Case (1 - Player Moves)

Summary: The game detects the player's movement in the real-world and generates content accordingly.
Priority: High
Extends: RunGame
Includes: Connection abruption
Participators: Actual player

	Actor	System
1	The player moves around in the real world	
2		The game detects the movement
3		The game updates the player's position on the map
4		The game updates the position of the existing players (and possibly lootboxes) relative to the player's new position (some players may disappear, some may show up)

Use Case (2 - Change Weapon)

Summary: The player can choose a weapon from their inventory.
Priority: Medium
Extends: RunGame
Participators: Actual player

	Actor	System
1	The actor clicks on the icon of the current weapon	
2		The system shows a list of available weapons, with their respective names, and specifications
3	User finds the desired weapon	
4	User chooses one weapon	
5		The icon for selected weapon is updated

Use Case (3 Buy item)
Summary: User chooses to buy a new item from the shop
Priority: Low
Extends: RunGame
Participators: Actual Player

	Actor	System
1	The user clicks on the shop icon	
2		System shows the items available for purchase.
3		Item information are also listed
4	User clicks on “buy”-button	
5		If the player can afford the item, the price of the item is

		deducted from the user's cash and item is added to the player's inventory
6	User can do step 4 again	
7	User exits the shop	
8		System returns to the last state of the game

Use Case 4 (Sell Item)
Summary: User chooses to one of its items to the shop
Priority: Low
Extends: RunGame
Participators: Actual Player

	Actor	System
1	The user clicks on the shop icon	
2		System shows the items available to sell.
3		Item information are also listed
4	User clicks on "Sell Item"-button	
5		The value of the sold item is added to the user's cash and item is removed from user's inventory
6	User can do step 4 again	
7	User exits the shop	
8		System returns to the last state of the game

Use Case (5 - Attack other player)
Summary: The user shoots at another player damaging them.
Priority: High
Extends: RunGame
Includes: Update Rank
Participators: Actual Player

	Actor	System
1	The player taps on the icon of an opponent	
2		System checks if the current weapon is NOT on cooldown
3		Damage from the weapon is calculated and applied on the opponent
4	Cooldown for the weapon starts ticking	
5	User can do step 1	
6		If the damage done by the player kills another player, the game updates the player's score and possibly rank as well.

Use Case (6 - Update Rank)
Summary: Player does an action that results in a change for the user's xp count, and thus possibly changing their rank, a change in rank can go in both directions i.e a promotion or demotion. The system should provide visual and feedback for the user, when they do something that changes their rank.
Priority: Medium
Extends: Attack Other Player, Get Killed
Participators: Actual Player

	Actor	System
1	User does an action that changes player's xp	
2		The system updates its stored information
3		The image for the rank changes to indicate the user's new rank

4		In case it's a promotion, the player is granted some gold

Use Case (7 - Get attacked)
Summary: The player takes damage from enemy combatant.
Priority: High
Extends: RunGame.
Includes: Getting Killed.
Participators: Actual Player, Enemy Combatant

	Actor	System
1	User takes damage from enemy opponent	
2		System calculates damage from the enemy and reduces the player's health
4		If damage sustained is fatal, user gets killed (see uc Get Killed)

Use Case (8 - Go offline/online-mode)
Summary: Player can switch the binary state of being visible to other players, which allows them to log out safely from the battlefield.
Priority: Medium
Extends: RunGame
Participators: Actual Player

Scenario 1:
Player is online and wants to go offline.

	Actor	System
1	User taps the radar icon to go offline-mode	
2		System checks to see whether the player can go offline
3.1 (player can go offline)		Player can no longer see anyone in their vicinity, attack, see lootboxes nor get attacked.
3.2(player cannot go offline)		A cooldown clock starts

		ticking down before the player can go offline

Scenario 2:

Player is offline and wants to go online

	Actor	System
1	User taps the radar icon to go online-mode	
2		Other players aswell as lootboxes are visible
3		Cooldown timer is started
4		Radar icon is disabled

Use Case (9 - Get killed)

Summary: Player's health bar has reached zero.
Priority: High
Extends: Get Attacked
Participators: Actual Player

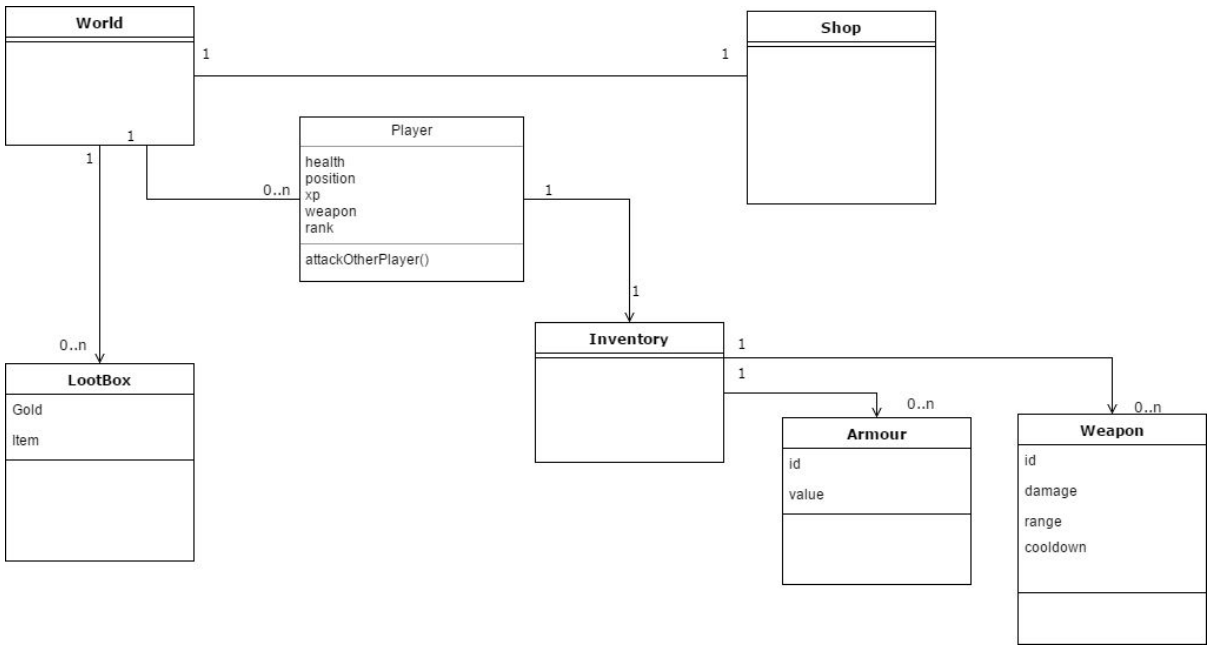
	Actor	System
1		Player becomes unable to do certain things, such as seeing other players or interact with them.
2		Player loses Exp and if necessary their rank changes (see uc 5- update rank)
3		After a set time, the player comes back to life reversing step 1.

Use Case (10 - Loot an item)

Summary: Pick up item dropped by killed opponent
Priority: Low
Extends: Attack other player, Player moves
Participators: Actual Player

	Actor	System
1	Moves toward displayed pile of loot on the map	
2	Taps the loot box to pick it up	
3		The loot disappears from the map
4		Looted items are added to player's inventory

4 Domain model An UML class diagram.



4.1 Class responsibilities

World: The world in which the game takes place.
LootBox: A box of loot which can include gold or items for the players to pick up.
Player: Represents the player of the game.
Inventory: The inventory of the player which can include Armours or Weapons.
Weapon: Weapon is the object representing how damage can be inflicted upon other players.
Armour: Armour decides how much damage the wielder will take upon getting shot by other players.
Shop: Shop is where a player can buy new items such as weapons and armours and sell the ones already owned.

5 Reference

https://developers.google.com/maps/documentation/android-api/