Proximity

Requirement and analysis document (RAD)

Version 6.0
Date 31/5 2015
Author: Linda Evaldsson, Simon Gislén, Johan Swanberg and Hanna Römer
This version overrides all previous versions

Contents

1. Introduction

1.1. Purpose of application

The purpose of this project is to create an entertaining game in the popular genre of Tower Defence. To differentiate from other tower defence games the game will have a bigger focus on direct user interactions. This will be accomplished by including a spell system. For information about the Tower Defence genre, see appendix 1.

1.2. General characteristics of application

The application will be a cross-platform desktop offline game runnable on Mac/Windows/Linux platforms. The game will be scalable and adapt to different screen sizes.

It will be a real time strategy (RTS) game based on the tower defence concept. It will be built of the basic components of a tower defence game: it will feature unlockable maps with creeps (enemies) approaching the player base. The player will place spells and towers that shoot projectiles to kill enemies and protect the base. The creeps will be generated in waves. These waves will continue to be generated until the player base has no more health or until the player has defeated a certain amount of waves to reach a win condition. Reaching the win condition on one map will unlock the next map in order. Devolving creeps will yield experience and resources (money).

The player will be able to choose a faction before starting a certain map. The player will always have four spells, but depending on what faction is chosen the player will have different kinds of spells at their disposal. The faction also affects the design of the base.
It is possible to level up the player. This is done automatically when experience is gained.

1.3. Scope of application

The application includes several unlockable playable maps with finished artwork, creeps, working towers and an end game scenario. A working system with some choosable factions that offer different spells in-game is also included. Between sessions the game will store the player data. This includes what maps are unlocked, the player level and the player experience.

1.4. Objectives and success criteria of the project

1. It should be possible to choose a map, buy towers with resources and cast spells. Enemies should be approaching on the map and if they reach the base it should lose health. Enemies should yield experience and resources when devolved.

2. The game should save the player data when it is closed.

3. The game should be relatively balanced, several strategies should be possible and viable to use.

1.5. Definitions, acronyms and abbreviations

- **TD**, tower defence.
- **Experience**, points gained from killing enemies. When a certain amount of experience is collected the next level is reached.
- **Level**, the player has a level. It is a display of how much experience the player has.
- **Wave**, the creeps come in clusters, the current wave is the cluster the player is currently facing.
- **Win condition**, a condition (for example a number of waves defeated) that the player needs to reach before winning.
- **Faction**, a theme with a spell-set that the player can choose from; different factions offer different spells.
- **Tower**, a block that the user can place on the screen. The tower costs resources. Most towers fire projectiles at enemies or attack them in some other way.
- **Projectile**, most towers can shoots projectiles that hit the enemies, these projectiles affect the creep(s) it hits, for example by freezing or devolving them.
- **Creep**, enemies that follow a certain path in the game and tries to kill the player. These can be of different types and strengths.
- **Devolve**, when a creeps dies, it "reincarnates" as a smaller creep type, if it's the smallest type of creep the creep gets destroyed.
- **Resource**, "money" that the player can use to purchase towers, different towers require different resources. The player gain resources by devolving creeps.
- **Line**, a type of resource.
- **Point**, a type of resource.
- **Polygon**, a more valuable type of resource.
- **Upgrade**, the player can upgrade towers to make them better
- **Spell**, the player gets four spells from the faction and this spell can be used to affect the game in the player's favor.

2. Requirements

2.1. Functional requirements

The player should be able to:

1. Start a new game
    a. Choose a map
    b. Choose a faction
2. Play the game
    a. Choose and place towers
    b. Open in-game menu
        i. Close in-game menu
    c. Get information about objects
    d. Go to the main menu
    e. Choose and place spells
    f. Change the game speed:
        i. Pause
        ii. Play (normal speed)
        iii. Fast forward
    g. Change settings
        i. Change the game music volume
        ii. Change the game effects volume
    h. Select tower
        i. Destroy selected tower
        ii. Change targeting method of selected tower
        iii. Upgrade selected tower
        iv. Deselect tower
3. Exit the application. The level, unlocked maps and experience will be stored.

The application should be able to:

1. Generate creep waves

2.2. Non-functional requirements

2.2.1. Usability

The game should not require more than a simple manual or a lot of previous experience to use except basic computer knowledge. A manual will be available on the game website (www.foh-proximity.se).

2.2.2. Reliability

Not applicable.

2.2.3. Performance

The interaction with the application should always seem instantaneous to the user.

2.2.4. Supportability

The application should be possible to run on Mac, Windows and Linux platforms. It should be easy to adapt the application to also run on a website (HTML5/Javascript) or an android or iOS device.

There should be automated tests for everything that is possible. For parts that cannot efficiently be tested automatically there will be manual tests. These will be included in Appendix 2. Testing of graphics rendering is not included as all graphics logic is supplied by a 3rd party.

2.2.5. Implementation

To make the application possible to run on all specified platforms the library LibGDX will be used. JVM (Java virtual machine) takes care of cross-platform compatibility.

2.2.6. Packaging and installation

The application will start from an executable file, no installation will be required. It will be possible to acquire the application on the application website (www.foh-proximity.se). If downloaded from the website a runnable .jar-file will be delivered. To run the application JVM (Java virtual machine) is needed.

2.2.7. Legal

The LibGDX library is distributed with the Apache2 license. The documentaton is not allowed to be obfuscated (libGDX, 2015).

The game will feature several songs as ingame music, this music will be chosen among songs licenced under the creative commons 3 licence (Creative Commons, 2015).

The term “tower defence” is trademarked. This term can’t be used in the game name in all countries, however the term can still be used as a reference to the game genre (Jordan, 2010).

2.3. Application models

2.3.1. Use case model

See appendix 3 for UML diagram of use cases and appendix 4 for textual descriptions.

2.3.2 Use cases priority

High priority use cases in priority order:

1. Run game

2. Exit
3. Pick up tower
4. Place tower
5. Put down tower

Medium priority use cases in priority order:

1. Change Speed
2. Pick up spell
3. Place spell
4. Put down spell
5. Choose map
6. Select tower
7. Deselect tower
8. Go to main menu

Low priority use cases in priority order:

1. Choose Faction
2. Upgrade tower
3. Open in-game menu
4. Close in-game menu
5. Change Music Volume
6. Change Targeting Method of tower
7. Destroy tower
8. Get information
9. Change Effects Volume

2.3.3 Domain model

See appendix 5 for the analysis model.

2.3.4 User interface

The design of the user interface will be based on a theme of geometrical figures. The game will be scalable and adapt to different screen resolutions. Se appendix 6 for the preliminary GUI.

2.4 References

Creative Commons. (2015). C*reative commons license.*
Retrieved from http://creativecommons.org/licenses/by/3.0/

libGDX. (2015). libgdx/libgdx.
Retrieved from https://github.com/libGDX/libGDX

Jordan, J. (2010). *Com2uS "guides" developers not to use its trademark Tower Defense.*
Retrieved from http://www.pocketgamer.biz/news/17997/com2us-guides-developers-not-to-use-its-trademark-tower-defense/

Tower defence. (2015). *In Wikipedia.*
Retrieved 2015-05-04 from http://sv.wikipedia.org/wiki/Tower_defense

APPENDIX

Contents

1. The tower defence genre

A tower defence game is a strategy game. The player decides where to position different kinds of towers on a map, and enemies will walk on this map to approach the players “home base”.

The strategy element consists of the player choosing which towers to buy, and where to position them. These towers often cost different amount of resources and have different effects, most often the towers shoot out projectiles on nearby enemies which cause damage or destroy the enemies.

The player loses health if the enemies reach the player base, as a result of the player not building a sufficient defence. A normal tower-defence round can last between 5-20 minutes, at that time the player has either died as a result of the approaching enemies, or has survived long enough that he has passed a limit which is considered a win-condition.

The player can then start a new round, where he will once again have to build up a sufficient defence.

2. Manual tests of the application

Some use cases are not possible to test automatically efficiently. These have instead been tested manually.

Facilitator: Linda Evaldsson
Date: 2015-05-30

2.1. Testing use case: Go to main menu

Entering a game and then going back to the main menu works. Both normal flow of events and the two alternate flow of events works.

2.2. Testing use case: Run game

Entering a game works as expected. The alternate flow of events where the player is on the lose screen and wants to restart the game works as expected.

3. UML diagram of use cases

Below you find an UML diagram of the use cases for Proximity.

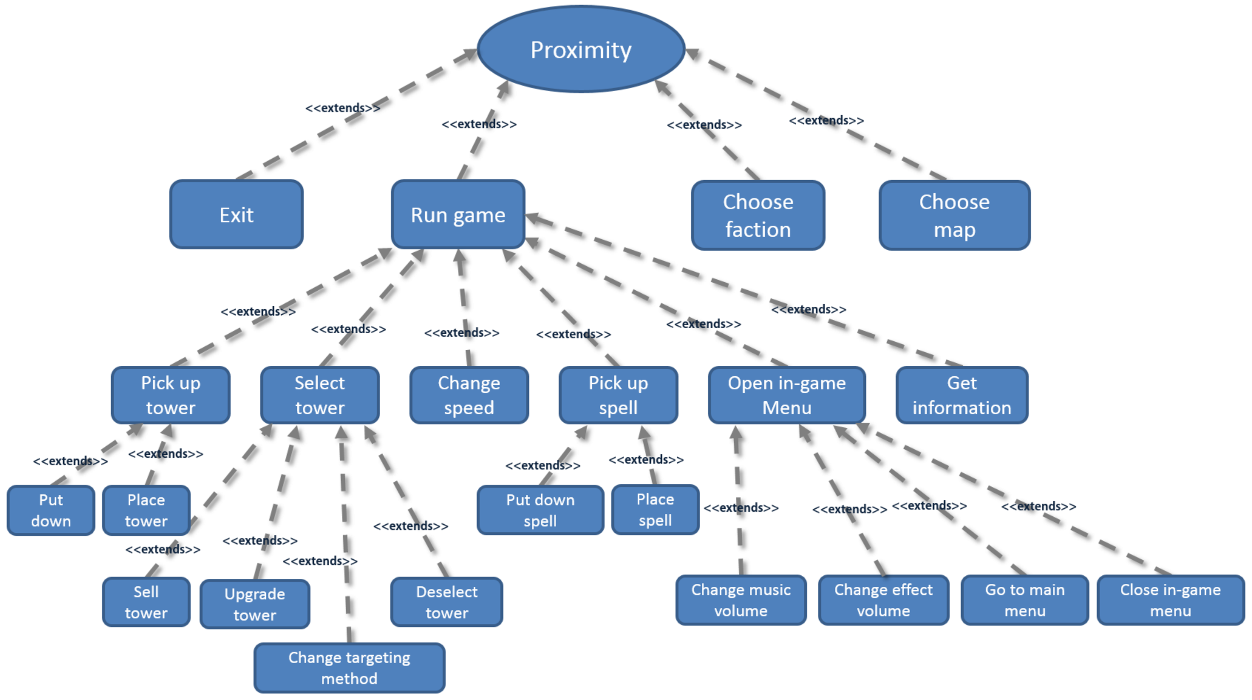


Figure 1: UML diagram of use cases for Proximity

4. Use case texts

Below all use cases for the application Proximity are found in alphabetical order.

List of use cases

4.1. Use case: Change effect volume

Summary:
This is how the effect volume in the game is changed.

Priority: Low
Extends: UC: Open in-game menu
Includes: -
Participators: Actual player

Normal flow of events
The player wants to change the effect volume

#	Actor	System
1	Player clicks on the effect volume bar corresponding to the effect volume the player wants	
2		Computer changes effect volume to the volume corresponding to the selected bar
3		The selected bar and all bars to the left of it are filled
4		All bars to the right of the selected bar are unfilled
5		The effect sound icon is shown as its normal icon (colored)

Alternate flow of events
Flow 2.1: The music is playing and the player wants to turn it off

#	Actor	System
2.1	Player clicks on the effect sound icon	
2.2		Computer changes the game effect volume to zero
2.3		All effect volume bars are shown as unfilled bars
2.4		The effect sound icon is displayed as a

		grayscale icon

Flow 3.1: The music is off and the player wants to turn it on to the same volume as it was before turning it off

#	Actor	System
3.1	Player clicks on the effect sound icon	
3.2		Computer changes the game effect volume to the volume that was before turning the sound off
3.3		The effect volume bar corresponding to the volume and all bars to the left of it are filled
3.4		All bars to the right of the corresponding bar are unfilled
3.5		The effect sound icon is shown as its normal icon (colored)

4.2. Use case: Change music volume

Summary:
This is how the music volume in the game is changed.

Priority: Low
Extends: UC: Open in-game menu
Includes: -
Participators: Actual player

Normal flow of events
The player wants to change the music volume

#	Actor	System
1	Player clicks on the music volume bar corresponding to the music volume the player wants	
2		Computer changes music volume to the volume corresponding to the selected bar
3		The selected bar and all bars to the left of it are filled
4		All bars to the right of the selected bar are unfilled
5		The music sound icon is shown as its normal icon (colored)

Alternate flow of events
Flow 2.1: The music is playing and the player wants to turn it off

#	Actor	System
2.1	Player clicks on the music sound icon	
2.2		Computer changes the game music volume to zero
2.3		All music volume bars are shown as unfilled bars
2.4		The music sound icon is displayed as a

		grayscale icon

Flow 3.1: The music is off and the player wants to turn it on to the same volume as it was before turning it off

#	Actor	System
3.1	Player clicks on the music sound icon	
3.2		Computer changes the game volume to the volume that was before turning the sound off
3.3		The music volume bar corresponding to the volume and all bars to the left of it are filled
3.4		All bars to the right of the corresponding bar are unfilled
3.5		The music sound icon is shown as its normal icon (colored)

4.3 Use case: Change game speed

Summary:
This is how the player changes the game speed; how fast the game flows.

Priority: Medium
Extends: UC: Run Game
Includes: -
Participators: Actual player

Normal flow of events
The player wants the game to play at normal speed

#	Actor	System
1	Clicks the normal speed-button	
2		Shows the normal speed-button as a “pressed down” button instead.
3		Other speed buttons are reset to the unpressed state.
4		Changes the game speed to normal speed.

Alternate flow of events
Flow 2.1: The player wants the game to play at high speed

#	Actor	System
2.1	Clicks the high speed-button	
2.2		Shows the high speed-button as a “pressed down” button instead.
2.3		Other speed buttons are reset to the unpressed state.
2.4		Changes the game speed to high speed.

Flow 3.1: The player wants to pause the game

#	Actor	System

3.1	Clicks the pause-button	
3.2		Shows the pause button as a “pressed down” button instead.
3.3		Other speed buttons are reset to the unpressed state.
3.4		Changes the game speed to paused state (zero speed).

Flow 4.1: The game is playing at normal or high speed and the player wants to pause the game

#	Actor	System
4.1	Presses the space bar	
4.2		Shows the pause button as a “pressed down” button instead.
4.3		Other buttons are reset to the unpressed state.
4.4		Changes the game speed to paused state (zero speed).

Flow 5.1: The player has paused the game and wants to restore the speed that was before the pause

#	Actor	System
5.1	Presses the space bar	
5.2		Shows the speed button (high-speed or normal-speed) corresponding to the speed that was before pausing as a “pressed down” button instead.
5.3		Other buttons are reset to the unpressed state.
5.4		Changes the game speed to the speed that was before pausing.

4.4. Use case: Change targeting method

Summary:
This is how the player changes the targeting method if a tower is selected that can target creeps.

Priority: Low
Extends: UC: Select Tower
Includes: -
Participators: Actual player

Normal flow of events
Player wants to change the targeting method

#	Actor	System
1	Player clicks on the check button on the tower panel corresponding to the targeting method the player wants the tower to have.	
2		The check button gets selected
3		Other check buttons on the tower panel gets unselected
4		The selected tower changes targeting method to the chosen method.

4.5. Use case: Choose faction

Summary:
This is how the Player chooses a faction to play as.

Priority: Low
Extends: -
Includes: -
Participators: Actual player

Normal flow of events
Player tries to do choose next faction

#	Actor	System
1	Player presses the "Next" button on the Faction Chooser	
2		The next faction in Faction Chooser's list of factions is displayed. If the current faction is the last in the list, then the first faction is displayed.

Alternate flow of events
Player tries to choose previous faction.

#	Actor	System
1	Player presses the "Previous" button on the Faction Chooser	
2		The previous faction in Factions Chooser's list of factions is displayed. If the current faction is the first in the list, then the last faction is displayed.

4.6. Use case: Choose map

Summary:
This is how the player chooses which map to play on

Priority: Medium
Extends: -
Includes: -
Participators: Actual player

Normal flow of events
Player tries to choose a map.

#	Actor	System
1	Player presses a map-icon for an unlocked map.	
2		The previous map-icon is set as Not selected.
3		The pressed map-icon is set as Selected.

Alternate flow of events
Flow 2.1: Player tries to choose a map that is not unlocked

#	Actor	System
2.1	Player presses a map-icon for a locked map.	
2.2		Computer does nothing.

4.7. Use case: Close in-game menu

Summary:
This is how the in-game menu is closed once it is open

Priority: Low
Extends: UC: Open in-game Menu
Includes: -
Participators: Actual player

Normal flow of events
Player tries to do something something

#	Actor	System
1	Player clicks the resume button	
2		Computer closes the in-game Menu
3		The speed is set to the same speed as it was before opening the menu

4.8. Use case: Deselect tower

Summary:
This is how the player unselects a tower.

Priority: Medium
Extends: UC: Run Game
Includes: -
Participators: Actual player

Normal flow of events
Player tries to deselect a tower.

#	Actor	System
1	Player clicks on a spot which does not trigger another selection, or presses the esc button.	
2		Clears any selected range indicators.
3		Clears the tower detail panel.

4.9. Use Case: Destroy tower

Summary:
This is how the player clears placed towers from the current map.

Priority: Low
Extends: UC: Select Tower
Includes: -
Participators: Actual player

Normal flow of events
Player tries to remove a tower from the map.

#	Actor	System
1	Player clicks on the “x” icon on the tower panel.	
2		Removes the selected tower from the map
3		Deselects the tower

4.10. Use case: Exit

Summary:
This is how the game is exited

Priority: High
Extends: -
Includes: -
Participators: Actual player

Normal flow of events
Player tries to exit the game

#	Actor	System
1	Player clicks on the regular X button to the top right or top left (depending on OS)	
2		The player data is saved; Map waves finished, player experience and player level.
3		The game is closed.

4.11. Use case: Get information

Summary:

This is how the player can get information about a certain item in the game.

Priority: Low
Extends: UC: Run Game
Includes: -
Participators: Actual player

Normal flow of events

Player tries to get information about an object in the game

#	Actor	System
1	Player hovers with the mouse over the object the player wants more information about.	
2		If the object has a box of information to display this information is showed in a transparent box that is created over the mouse position.
3	Player is done with the information and places the mouse outside the transparent box.	
4		The box of information is closed.

4.12. Use case: Go to main menu

Summary:
This is how the player can return to the main menu from a game.

Priority: Medium
Extends: UC: Open in-game Menu
Includes: -
Participators: Actual player

Normal flow of events
Player tries to go to the main menu. The player has passed 40 waves on the map.

#	Actor	System
1	Player presses Main menu-button on the in-game menu.	
2		The screen displays the main menu.
3		The map is added to the player's list of won maps.
4		If the number of waves is greater than the record for the map, its map-icon is updated to display the new record.
5		The map after this one's map-icon is set as selectable.

Alternate flow of events
Flow 2.1: Player tries to go to the main menu. The player has not passed 40 waves on the map.

#	Actor	System
2.1	Player presses Main menu-button on the in-game menu.	
2.2		The screen displays the Main Menu.
2.3		If the number of waves is greater than the record for the map, its map-icon is updated to display the new record.

Flow 3.1: Player is on the losing screen and wants to go to the main menu.

#	Actor	System
2.1	Player presses Main menu-button on the losing screen.	
2.2		The screen displays the Main Menu.
2.3		If the number of waves is greater than the record for the map, its map-icon is updated to display the new record.

4.13. Use case: Open in-game menu

Summary:
This is how the player can open the in-game menu while playing the game.

Priority: Low
Extends: UC: Run Game
Includes: -
Participators: Actual player

Normal flow of events
Player tries to open the menu

#	Actor	System
1	Player clicks on the gear-icon	
2		The in-game Menu is displayed in the middle of the screen.
3		The game speed is paused

4.14. Use case: Pick up spell

Summary:
This is how the player picks up a spell while playing the game.

Priority: Medium
Extends: UC: Run Game
Includes: -
Participators: Actual player

Normal flow of events
Player picks up a spell.

#	Actor	System
1	Player clicks on the spell that is wanted on the spell panel or presses a corresponding key.	
2		An indicator is spelled around the mouse that shows the range of the spell in a green color.

Alternate flow of events
Flow 2.1: Player picks up a spell which has not yet finished its cooldown.

#	Actor	System
2.1	Player clicks on the spell that is wanted on the spell panel or presses a corresponding key.	
2.2		An indicator is spelled around the mouse that shows the range of the spell in a red color.

4.15. Use case: Pick up tower

Summary:
This is how the actual player places picks a tower up.

Priority: High
Extends: Run game
Includes: -
Participators: Actual player

Normal flow of events
Player tries to pick up a tower.

#	Actor	System
1	Clicks on the tower icon on the purchase panel or presses a corresponding key.	
2		Mouse cursor turns into an indicator showing an icon of the tower which indicates where the tower will be placed and the range the tower would cover if placed. What color the range is displayed in depends on where the mouse is hovering (se use case: place tower).

4.16. Use Case: Place spell

Summary:
This is how the player places a spell on the map.

Priority: Medium
Extends: UC: Pick up spell
Includes: -
Participators: Actual player

Normal flow of events
Player tries place a spell on the map.

#	Actor	System
1	Player clicks on map.	
2		Computer creates a spell on the cursor location.
3		Computer unselects (put down) the current selected spell.
4		Computer starts a spell cooldown & cooldown indicator.

Alternate flow of events
Flow 2.1: Player tries to cast a spell but the spell has not finished its cooldown.

#	Actor	System
2.1	Player clicks on the map.	
2.2		Computer unselects (put down) the current selected spell.

4.17. Use case: Place tower

Summary:
This is how the actual player places a tower on the map.

Priority: High
Extends: Pick up tower
Includes: -
Participators: Actual player

Normal flow of events
Player tries to place a tower on the map.

#	Actor	System
1	Hovers the mouse over the map where the tower should be placed	
2		Mouse cursor shows an indicator showing an icon of the tower which indicates where the tower will be placed and the range the tower would cover if placed. The range is displayed in green.
3	Clicks to place the tower	
4		A new tower is constructed on the spot where the player clicked.
5		The system reacts as if the user selected the tower, see Use case: Select Tower

Alternate flow of events
Flow 2.1: Player tries to place a tower on the map but has insufficient funds.

#	Actor	System
2.1	Hovers the mouse over the map where the tower should be placed	
2.2		Mouse cursor shows an indicator showing an icon of the tower which indicates where the tower will be placed and the range the tower would cover if placed. The range is displayed

		in red.
2.3	Clicks to place the tower	
2.4		Tower is not placed.

Flow 3.1: Player tries to place a tower on a path on the map

#	Actor	System
3.1	Hovers the mouse over the path where the tower should be placed	
3.2		Mouse cursor shows an indicator showing an icon of the tower which indicates where the tower will be placed and the range the tower would cover if placed. The range is displayed in yellow.
3.3	Clicks to place the tower	
3.4		Tower is not placed.

4.18. Use case: Put down spell

Summary:

This is how the player unselects a spell which has not yet been placed.

Priority: Medium
Extends: UC: Pick up spell
Includes: -
Participators: Actual player

Normal flow of events

Player tries to put down a spell that is held in the hand

#	Actor	System
1	Player clicks on a control panel or presses “esc”.	
2		Computer unselects the currently chosen spell.

4.19. Use case: Put down tower

Summary:

This is how the player unselects a tower chosen from the purchase-panel.

Priority: High
Extends: UC: Pick up Tower
Includes: -
Participators: Actual player

Normal flow of events

Player selects a tower from the purchase menu, but then changes his mind and unselects it.

#	Actor	System
1	Player clicks on the control-panel or presses “esc”.	
2		The current tower is unselected (“put down”)

4.20. Use case: Run game

Summary:
This is how the player starts a game.

Priority: High
Extends: Choose faction, Choose map.
Includes: -
Participators: Actual player

Normal flow of events
Player tries to start a new game.

#	Actor	System
1	Player presses the start game button.	
2		The selected map is loaded.
3		The selected factions spells are loaded.
4		The player acquires starting resources for buying towers
5		The system starts generating waves of enemies that moves towards the player base

Alternate flow of events
Flow 2.1: Player has lost the game and wants to restart and play the same map again

#	Actor	System
2.1	Player presses the resume button	
2.2		The map that last was played is loaded
2.3		Cooldowns of factions spells are reset
2.4		The player acquires starting resources for buying towers
2.5		The system starts generating waves of enemies that moves towards the player base

4.21. Use case: Select tower

Summary:

This is how the player chooses a placed tower on the map for further interaction and range indication.

Priority: High
Extends: UC: Run Game
Includes: -
Participators: Actual player

Normal flow of events

Player tries to select a tower on the map.

#	Actor	System
1	Player clicks on a tower on the map	
2		Shows a blue circle indicating the towers range unless range is infinite
3		Shows the tower detail panel with available upgrades and targeting methods.

4.22. Use case: Upgrade tower

Summary:
This is how the player upgrades a selected tower.

Priority: Medium
Extends: UC: Select Tower
Includes: -
Participators: Actual player

Normal flow of events
Player tries to upgrade the selected tower.

#	Actor	System
1	Player presses Upgrade-button or "F" on the keyboard.	
2		The tower on the map is replaced with the upgraded version.
3		The amount of resources the upgrade cost is removed from the player's resources.
4		The Selected-tower display now shows the upgraded version

Alternate flow of events
Flow 2.1: Player tries to upgrade the selected tower, but has insufficient resources for the cost of the upgrade.

#	Actor	System
2.1	Player presses Upgrade-button or "F" on the keyboard.	
2.2		Computer does nothing.

5. Preliminary analysis model

This is the preliminary analysis model for the application Proximity.

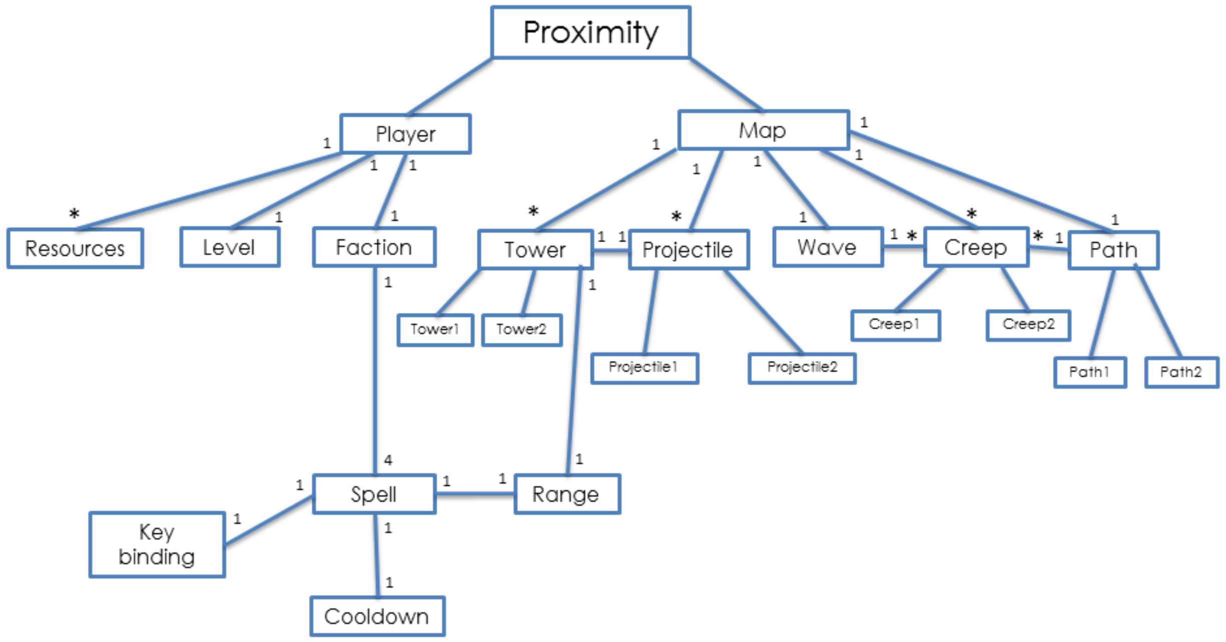
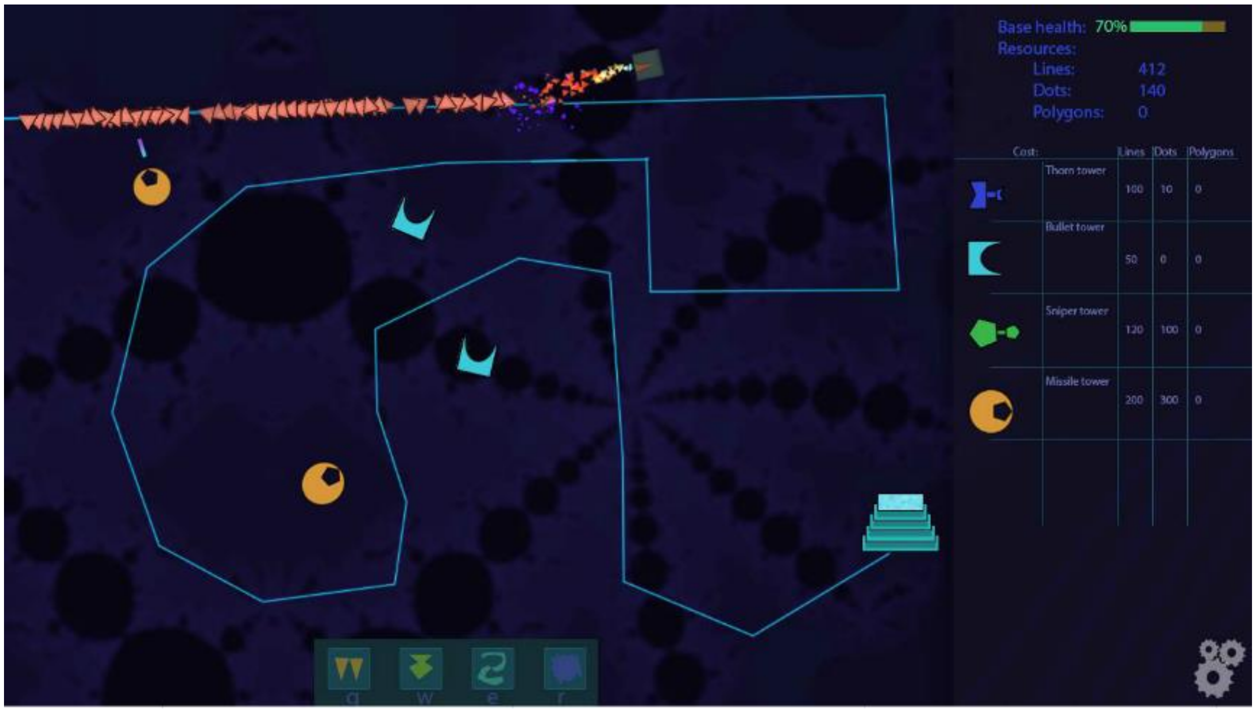


Figure 1: Preliminary analysis model

6. Preliminary GUI

This is the preliminary graphical user interface for the application Proximity.



Base health: 70%
Resources:
Lines: 412
Dots: 140
Polygons: 0

Cost:		Lines	Dots	Polygons
	Thorn tower	100	10	0
	Bullet tower	50	0	0
	Sniper tower	120	100	0
	Missile tower	200	300	0

Figure 1: Preliminary Graphical User Interface