Bilkent University

Department of Computer Engineering

Senior Design Project

Project Analysis Report

Project Name: Espionage

Project Group Members:

Özgür Öney	21101821
Ahmet Safa Kayhan	21001532
Selçuk Gülcan	21101231
Ateş Balcı	21202771
Fırat Özbay	21201683

Supervisor: Can Alkan

Project Website: espionage-game.github.io

Project Analysis Report
November 2, 2015
This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Project CS491/2

Contents

1. Introduction 3
2. Proposed System 4
2.1 Overview 4
2.2 Functional Requirements 4
2.3 Non-Functional Requirements 5
2.4 Pseudo Requirements 5
2.5 System Models 6
2.5.1 Use Case Model 6
2.5.2 Object and Class Model 8
2.5.3 Dynamic Models 9
2.5.4 Screen Mock-ups 11

1. Introduction

Video games that get involved our lives in pre 60's, from these day to nowadays confront us as not only a basic show business element but also software that is used by people from every age and worth millions and billions worth market share. Such software that plough through software world with the help of devices called game consoles at the very beginning became very popular just after personal computers proliferation such that there is at least one PC in every single house.

Espionage, basically is an two dimensional (2D) stealth-arcade game which has character left into a map to go from one point to another with the help of supplied inventory and where users directs this character. Map, that player will follow to finish the particular level is not constant structure of course; it will be designed as layered and will be looking like a lateral section of a building. In addition to its lateral section structure, it will contain elements that have different characteristics; such as boxes will be set down to hide behind them, stairs will be there to elevate between floors/layers, vent holes to pass through enemies without getting attention etc. Needless to say that, number of elements will differ from level to level with the help of machine learning algorithm described below to change difficulty.

On player's way to the end, computer-controlled units that have own adaptive intelligence attempts to fail him whereas player attempts to complete the game by reacting with reflexive and strategic moves. As character is controlled by player, enemy units that aim to fail player will be controlled by artificial intelligence algorithm, which means that they will also behave in natural ways to give reaction to player's character. For example, whenever character is seen by an enemy unit, such enemy unit will alert all other enemy as well as change their constant formation (such as patrol route or condition of being armed etc.). In addition to that, a machine learning algorithm will detect the player's style by gathering data like time spent, number of tools used or number of enemies neutralized. Such processed data will be used to determine the level of difficulty of course, hereby play a critical role on playability and enjoyment.

Most importantly, a genetic algorithm will be implemented for the game. As the name suggest, a spectrum basic artificial intelligences for every enemy unit will be implemented first and game will be opened to many players to play. As time goes and player's play the game to finish, just enemy units that have adequate intelligence will be survived, while others were simply eliminated. To do this, of course a score mechanism for enemy units to detect ones those are able to survive more. At last, a combination of surviving ones will be created to construct a sound basis artificial intelligence.

2. Proposed System

2.1 Overview

The project requirements and models are defined for the implementation of the system as well as proposed machine learning algorithm and artificial intelligence algorithms. Requirements are divided into three sub headings respectively and given first. Right after, use case diagram given below will describe the user activities just after game started with basic demonstrative scenarios, while sequence diagrams shows the background system activities – interaction of modules or classes. However, when we look at the class diagram, we can see a basic class structure of "enemy" units in game, since other classes are still not envisaged. As the times goes by, additional information will be given for other modules, of course. At last, mock-ups from main menu will be provided, to increase

2.2 Functional Requirements

- ✓ The game will be controlled by mouse and keyboard.
- ✓ The game will be displayed on a visual display and will be in preferred resolution.
- ✓ The game will feature music and sound effects; however these will not be crucial for gameplay.
- ✓ Speakers or headphones are an optional user requirement.
- ✓ The game must have a title screen with buttons that allow navigation to the game screen, instructions screen and credits screen.
- ✓ The game must have an instructions screen.
- ✓ The user can navigate from the instructions screen back to the title screen.
- ✓ The game will have a credits screen.
- ✓ The user can navigate from the credits screen back to the title screen.
- ✓ The game will feature 'next level' screens that appear between the levels of the game. These screens show the current score. A button allows navigation to the next level.
- ✓ The player character can move up, down, left and right, using the arrow keys.
- ✓ The game will have more than one level. Each level must be completed within a time limit or the game will end. Time limit changes from one level to another and also determined with help of Machine Learning algorithms.
- ✓ The player character must achieve assigned mission and reach the trapdoor within the time limit to proceed to the next level. The player can pick of Easter Eggs hidden in the game.
- ✓ Easter Eggs includes different character outfits, hidden messages audial or visual add-ons to title screen.

- ✓ The game ends when the player character completes the last level.
- ✓ Scoring in the game is based on how much time spent to complete a level.
- ✓ The current score and level are displayed on the game screen.
- ✓ The game will feature appropriate sound effects. A sound will play when the player neutralize an enemy. A sound will play when the player uses an inventory item. A 'victory' sound will play when the player successfully completes a level.
- ✓ Appropriate music will play throughout the game.

2.3 Non-Functional Requirements

- ✓ The minimum frame rate (FPS) must be 40-50 per second.
- ✓ Average frame rate (FPS) must be approximately 60 per second.
- ✓ The average response time between click and reaction must be less than 0.5 seconds. The maximum response time between click and reaction must be two seconds. Adding some simple classes and methods that will compute and display the time needed to process any operation can test this requirements.
- ✓ The game must be able to run with minimum 1024 megabytes of RAM.
- ✓ The game must run in Windows. To verify this requirement, installing the game into appropriate environment and check whether it works properly or not would be sufficient.
- ✓ Code written must be maintainable. This can be achieved by collecting metrics, such as DIT (depth in inheritance tree), MPC (message-passing coupling), WMC (weighted method complexity) and DAC (data abstraction coupling). Also adding documentation will improve the maintainability scale of the system.

2.4 Pseudo Requirements

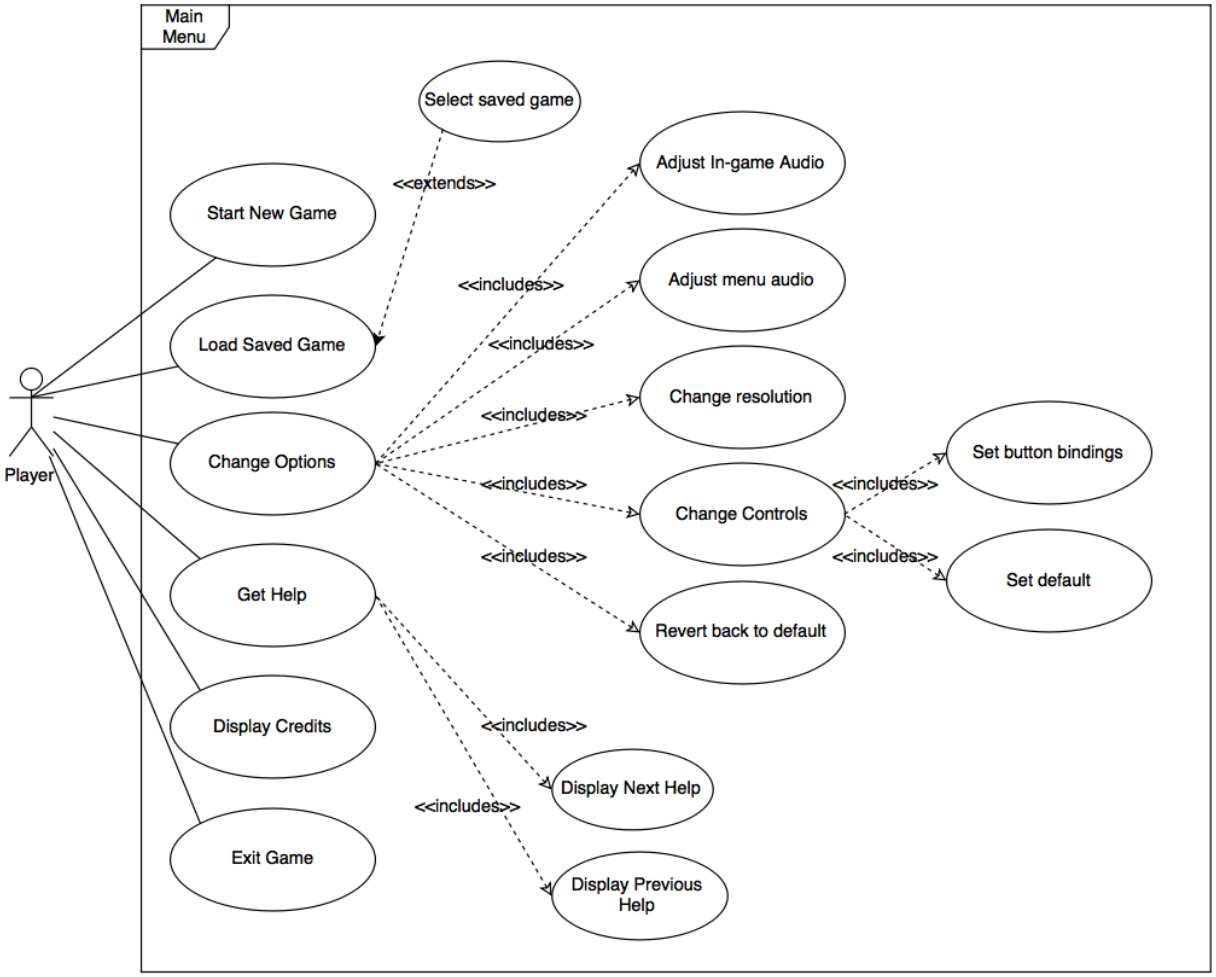
- ✓ The game will be implemented in C#.
- ✓ The machine learning algorithms will be implemented in Python.
- ✓ The application will be implemented for PC.
- ✓ Object oriented design principles will be applied for whole project.
- ✓ In-game sound will be changing from one level to another.
- ✓ Visual details will be changing from one level to another.
- ✓ Animations for enemy and player character will be designed in Adobe Illustrator.
- ✓ To increase reality, a physics engine will be used.
- ✓ At very first, boundaries will be shown in a tutorial gameplay.
- ✓ Help screen will demonstrate boundaries clearly.
- ✓ User interface will be designed to reflect game's main idea.

2.5 System Models

Roles of users and all other units (which are under control of artificial intelligence algorithms) will be defined in subheadings below, with the help of Unified Modeling Language diagrams.

2.5.1 Use Case Model

In following use case diagram, activities which can be done in *main menu* of the game by user is described. In addition to that, descriptions are written just after the diagram.



Use Case Name: Start New Game
Participating Actors: Player
Entry Condition: None.
Flow of Events:
1. User opens the game main menu
2. User clicks to “Start New Game” button
Exit Conditions:
1. User clicks to “Back to Main Menu” button

Use Case Name: Load Game
Participating Actors: Player
Entry Condition: None.
Flow of Events:
1. User opens the game main menu
2. User clicks to “Load Game” button

Exit Conditions:
1. User clicks to “Back to Main Menu” button

Use Case Name: Select Saved Game
Participating Actors: Player
Entry Condition: Player clicks Load game button
Flow of Events:
1. User opens the game main menu
2. User clicks to “Load Game” button
3. User selects between list of previously saved games
4. User clicks on his decide.
Exit Conditions:
1. User clicks to “Back to Main Menu” button

Use Case Name: Change Options
Participating Actors: Player
Entry Condition: None.
Flow of Events:
1. User opens the game main menu
2. User clicks to “Change Options” button
Exit Conditions:
1. User clicks to “Back to Main Menu” button

Use Case Name: Adjust Audio
Participating Actors: Player
Entry Condition: User clicks on “Change Options” Button
Flow of Events:
1. User opens the game main menu
2. User clicks to “Change Options” button
3. User adjust in-game or menu audio volume with the help of bar
Exit Conditions:
1. User clicks to “Back to Main Menu” button

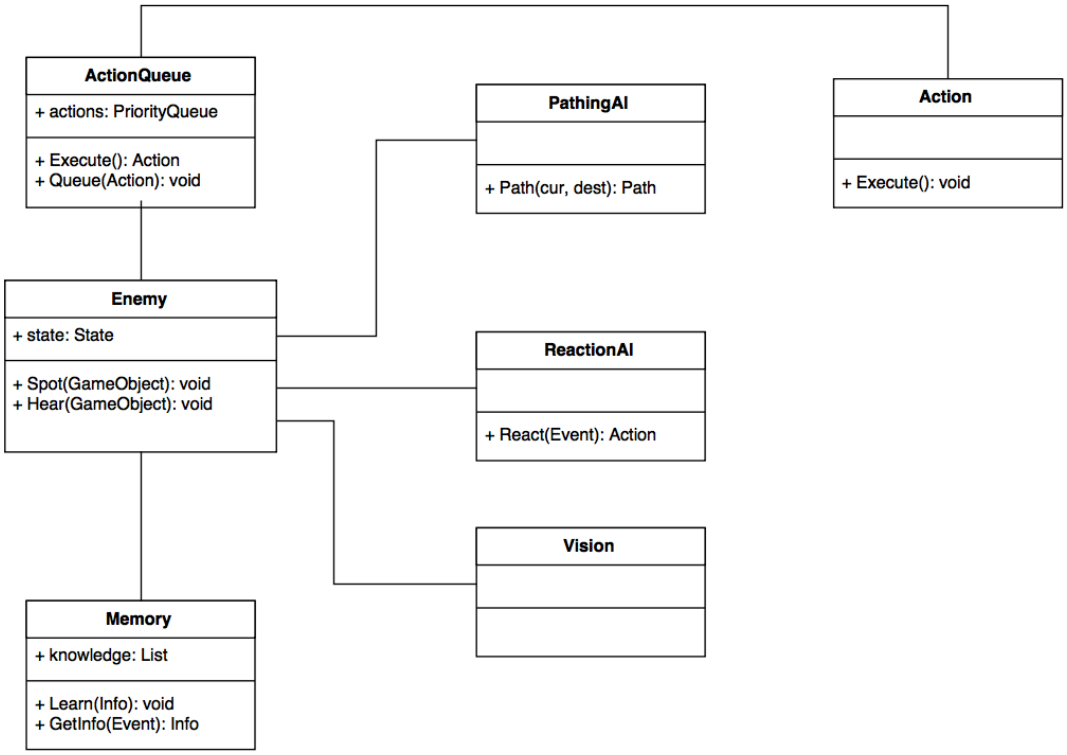
Use Case Name: Change Resolution
Participating Actors: Player
Entry Condition: User clicks on “Change Options” Button
Flow of Events:
1. User opens the game main menu
2. User clicks to “Change Options” button
3. User adjust resolution by selecting pre-defined resolutions from developer.
Exit Conditions:
1. User clicks to “Back to Main Menu” button

Use Case Name: Set button bindings
Participating Actors: Player
Entry Condition: User clicks on “Change Options” Button and just after that clicks on “Change Controls”
Flow of Events:
1. User opens the game main menu

2. User clicks to “Change Options” button
3. User clicks to “Change Controls” button.
4. User changes buttons as he wants
Exit Conditions:
1. User clicks to “Back to Main Menu” button
2. User clicks to “Set default” button and navigates through Main Menu

Use Case Name: Get Help
Participating Actors: Player
Entry Condition: None.
Flow of Events:
1. User opens the game main menu
2. User clicks to “Get Help” button
Exit Conditions:
1. User clicks to “Back to Main Menu” button
2. User displays all help screen items.

2.5.2 Object and Class Model

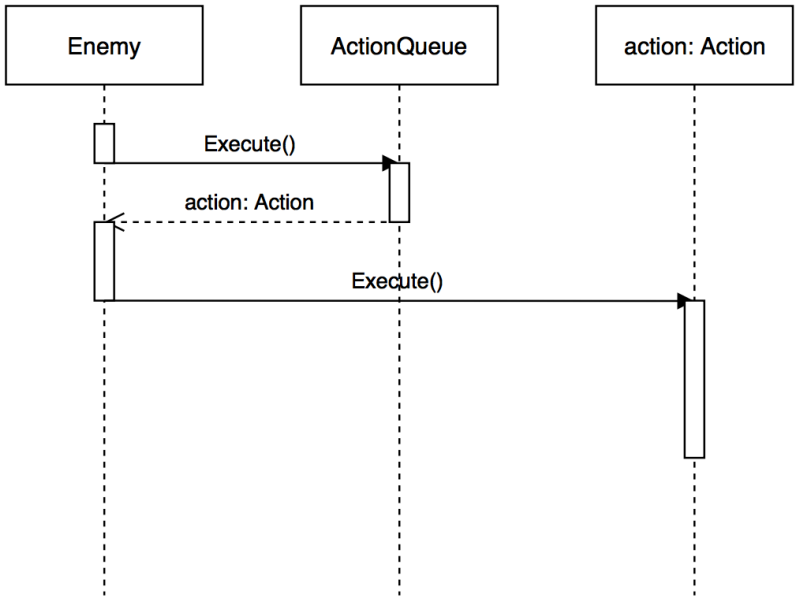


Since we have focused on enemy and artificial intelligence of enemy, we firstly sketched enemy class diagram. However, as project progress, other classes will be detailed.

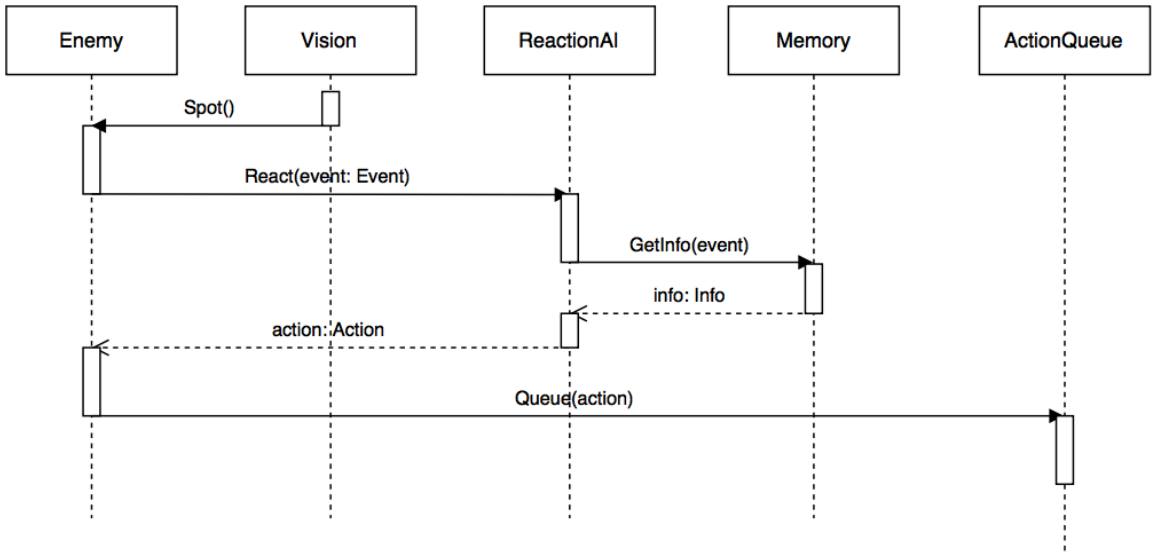
2.5.3 Dynamic Models

2.5.3.1 Sequence Diagrams

Following sequence diagrams explains what really happens among classes while their respective scenario happens.

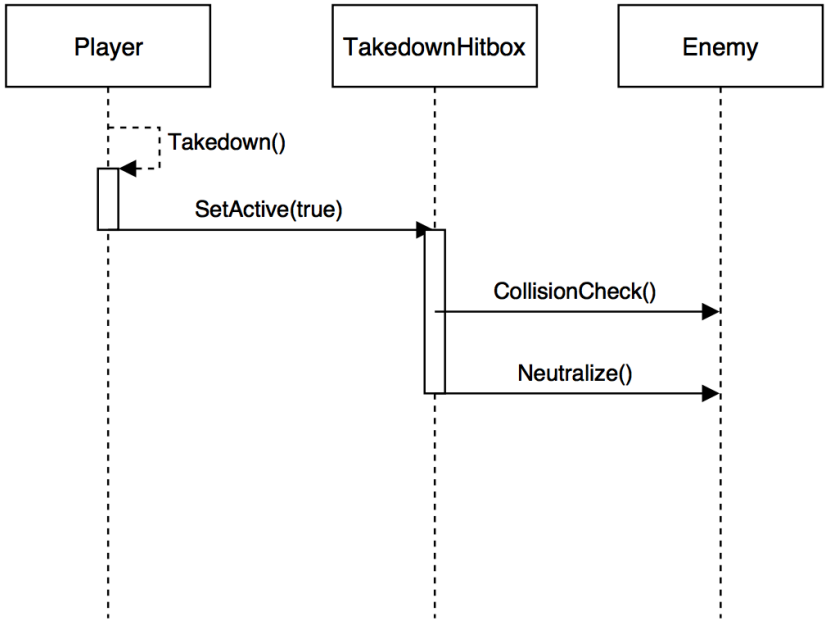


Scenario name: Enemy Action Execution
Actors: AI Unit
Flow of Events:
1) Enemy looks up the next action from the queue
2) Retrieves the action
3) Behaves according to the said action

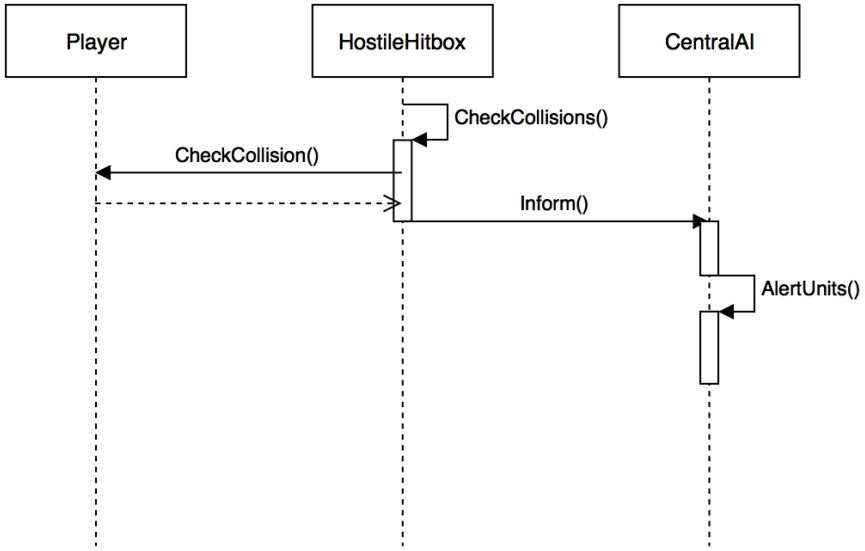


Scenario name: Enemy Spots an Object
Actors: AI Unit
Flow of Events:
1) Enemy spots the object
2) Calls the reaction AI component
3) Reaction component looks up previous memory to determine a course of action

4) Retreives the info from memory component
5) Determines the course of action
6) Sends the action back to the main Enemy class
7) Enemy adds the action to the action queue



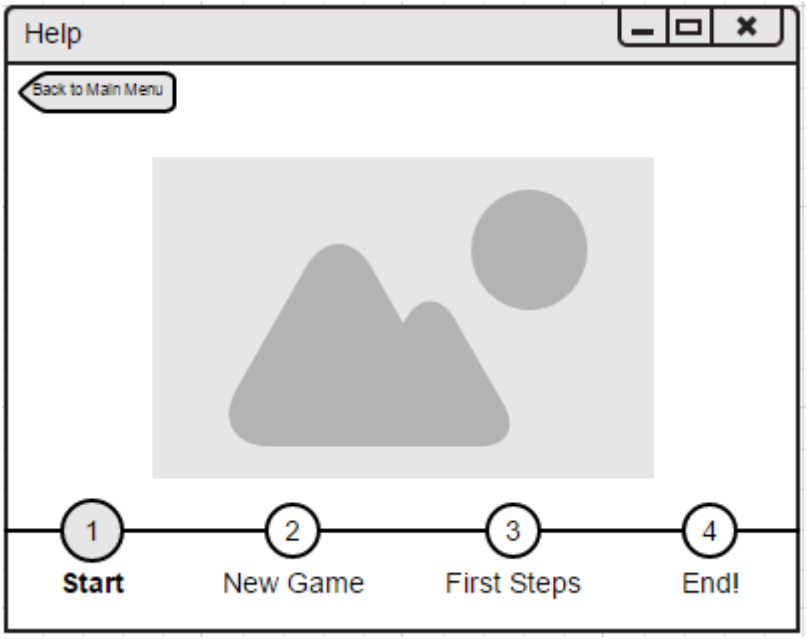
Scenario name: Player knocks out an Enemy
Actors: User, AI Unit
Flow of Events:
1) User controlled character is faced with an enemy
2) User controlled character approaches the enemy without being detected
3) User controlled character is given an input to takedown the enemy
4) Player class creates a collision box
5) Collision box checks whether it contains an enemy unit or not
6) Necessary animation for takedown is played
7) Existent enemy object is neutralized



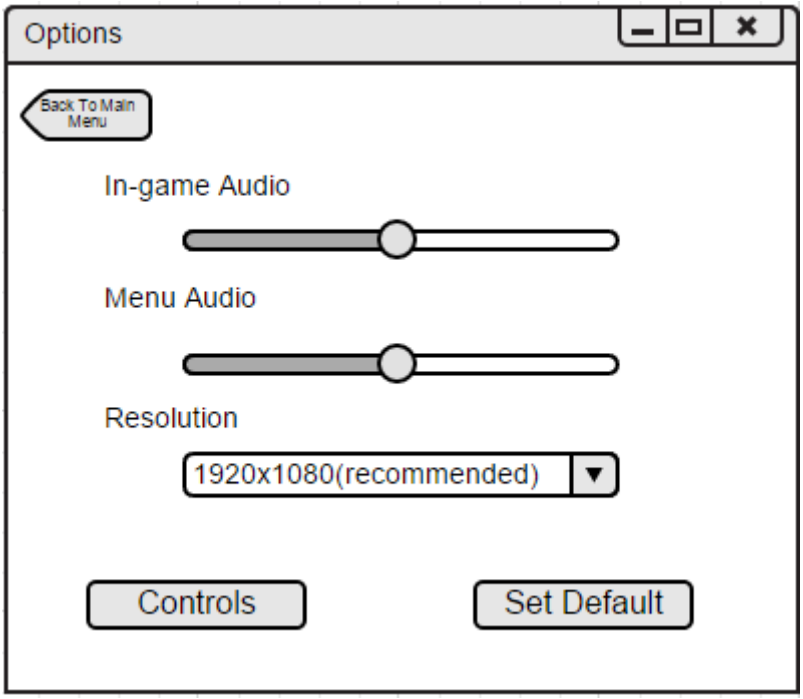
Scenario name: Detection of Player
Actors: User, Enemy AI
Flow of Events:

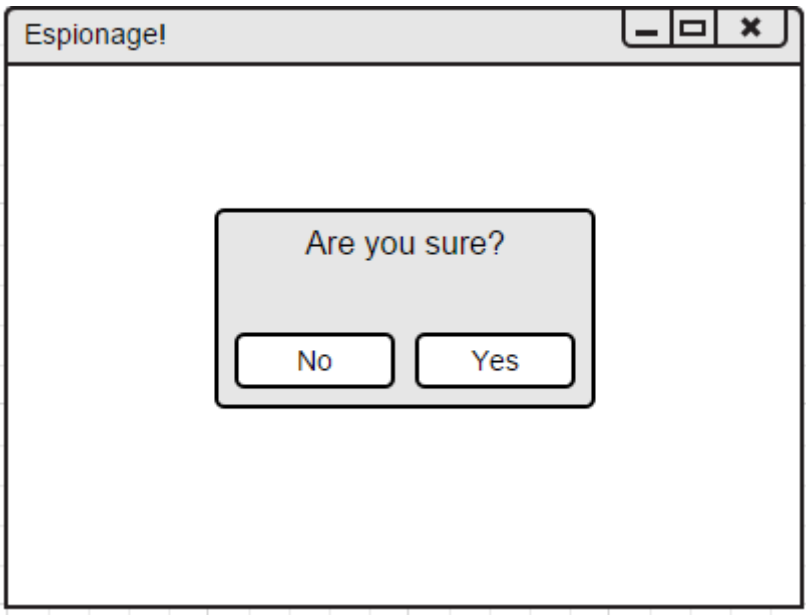
1) User controlled character is guided to an alert zone
2) While user moves, hostile object scans the area
3) Alert is triggered by its collider
4) AI unit informs the rest of the hostile units of the players presence
5) All units enters an alert state and behaves accordingly

2.5.4 Screen Mock-ups


Help
Back to Main Menu
1
Start
2
New Game
3
First Steps
4
End!


Load Game
Back To Main Menu
Saved Game 1
Saved Game 2
Saved Game 3
Saved Game 4
Saved Game 5


Options
Back To Main Menu
In-game Audio
Menu Audio
Resolution
1920x1080(recommended)
Controls
Set Default


Espionage!
Are you sure?
No
Yes