|  |
| --- |
| UAS Software Development Project 1 |
| Ninja Game |
| Software Design Document |

|  |
| --- |
| Team members:  Dair Baidauletov  Fayjus Salehin  Hung Dao |

Contents

[1. Introduction 2](#_Toc479520964)

[1.1 Scope of the Project 2](#_Toc479520965)

[1.2 Document Overview 2](#_Toc479520966)

[1.3 Reference Material 2](#_Toc479520967)

[1.4 Definitions and Acronyms 2](#_Toc479520968)

[2. User Interface 2](#_Toc479520969)

[2.1 Site Map 2](#_Toc479520970)

[2.2 Page Layout and Design 3](#_Toc479520971)

[3. Client Side SW Design 4](#_Toc479520972)

[3.1.1 Framework 4](#_Toc479520973)

[3.1.2 Classes 4](#_Toc479520974)

[3.1.3 Animations 6](#_Toc479520975)

[4. Database 6](#_Toc479520976)

[5. Server Side SW Design 6](#_Toc479520977)

# Introduction

## Scope of the Project

We are making a mini game for entertainment purposes. “Ninja game” is a simple web based 2D endless running game[1] with a database system that allows player to track top scores. The game is using modern internet technologies and is capable to be played by people all over the world.

## Document Overview

This document describes the main design basis of our project. Our team consists of motivated students diving into a software development process for the first time. The interactive mini-games are easy way to connect with people and they do not rely hardly on the social interests and age groups, which makes the product universal and omnibus.

## Reference Material

[1] — en.wikipedia.org/wiki/Platform\_game

[2] — https://p5js.org

[3] — https://en.wikipedia.org/wiki/Sprite\_(computer\_graphics)

[4] — https://www.w3schools.com/xml/ajax\_intro.asp

## Definitions and Acronyms

**Class** — main unit of the object-oriented programming representing the inheritable and extendable set of variables and functions.

**Client side** — a set of processes that are executed on user’s program (browser).

**Server side** — a set of processes that are executed on servers.

AJAX — Asynchronous Javascript and XML, is a way to communicate with server-side scripts.

JS — Javascript, the internet client side scripting language.

MySQL — Structured Query Language, domain-access language to access relational database.

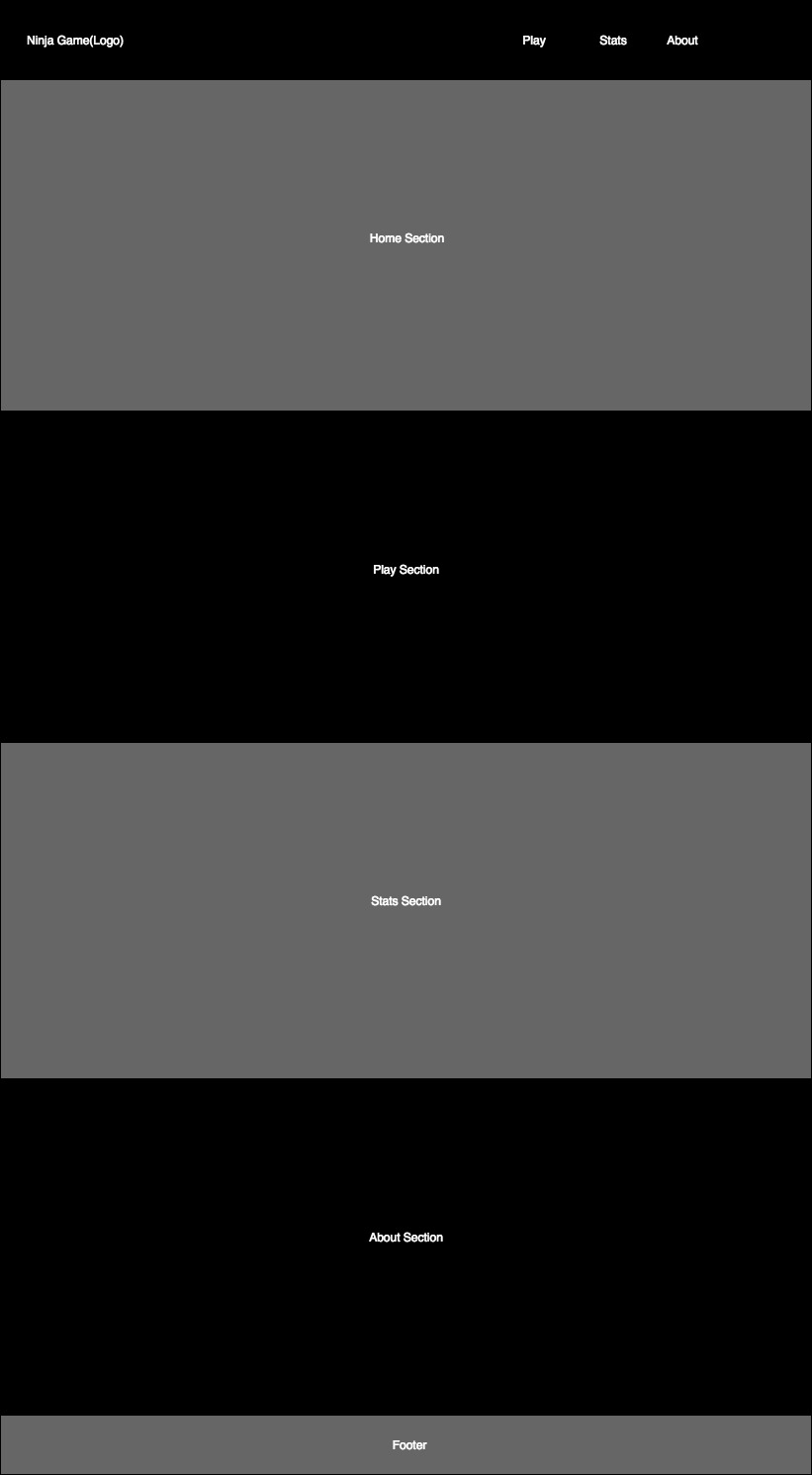
PHP — Hypertext preprocessor, client side scripting language.

# User Interface

## Site Map

Ninja Game is a single page scrolling web site. Instead of having multiple pages, it has a single *.php* page where sections are divided into <div> elements. The website has four different sections – **Home**, **Play**, **Stats** and **About**. The user can switch to different sections by clicking the section name from the navigation bar (like a regular multi page website) or scroll down through the section.

## Page Layout and Design



# Client Side SW Design

## 3.1.1 Framework

After couple of experiments of using the native HTML rendering and animation techniques, we found out that in order to make the game playable, we should use some rendering *Javascript* framework. Our choice fell on modern **p5.js[2]** framework and using it makes such nuances as smooth animation, easy controllers, audio background etc. not exceedingly complicated.

The framework requires some preloading and setup methods to handle variable declaration, assets attachment and assignment operations. The next *draw()* function is autonomously called once per rendered frame and it keeps tracking of all the processes and occurrences in code with possible *if* constructs. This helps us to concentrate on actually making game mechanics and loop flow work properly and with no extra exceptions or errors.

## 3.1.2 Classes

There are 9 javascript classes in the source code — 1 Ninja class for player , 1 abstracts class Collider, 2 subclasses of Collider — Enemy and Pickup, and actual 3 children of Enemy (Shuriken, Kunai, Katana) and 2 of Pickup (Health, Coin).

C:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Untitled Diagram (2).png

The reason for creating an abstract class Collider is in optimization of code. As it is all of the collider shall have *move(), show()* and *crash()* methods, therefore it is more convenient to put them into an abstract class first.

The reason for creating two different Enemy and Pickup classes, although they require similar value of integer, is in creating a logical distinction in uses of mentioned classes. Therefore, the Enemy classes affect the health of Ninja (deteriorates), whereas Pickup may perform any other interactions.

* **Ninja** is the main controllable player-object.

Data fields:

*(int) height, width, x-position, y-position, healt, score*

Member functions:

*move()* and *show()* functions are called in main loop and are respectively responsible for moving the coordinates according to player inputs and displaying the corresponding animation of ninja. The move function also sets the boundaries to the movement of player, illusion of ground and gravity imitation.

* **Collider** is an abstract class created for being extended by class **Enemy** and **Pickup**. The reason for doing so is to optimize code by not repeating similar functions. When created it first randomizes the position of object.

Data fields:

*(int) x-position, y-position, speed, width, height*

Member functions:

*move()* — simple arithmetic function for changing the *x* coordinate according to *speed* parameter;

*crash( Ninja )* — takes the *Ninja* object as a parameter, checks for collision to occur and performs actions for any of particular case by checking the type of an object with ‘*instanceof’* command. Therefore, if player have crashed into a *Coin*, his score is increased, if crashed into *Katana* his health is decreased by a certain value.

* **Enemy** (inherits Collider) is used for imitating obstacles and decreasing Ninja object’s health field when colliding.

Data fields:

All data fields inherited from super class *Collider.* The new data field *damage* is an integer value which is decremented from the player’s health when crash occurs.

Member functions:

All functions inherited from super class *Collider*.

* **Pickup** (also inherits *Collider*) is used for rising the score/health of the object Ninja when colliding with it.

Data fields:

All data fields inherited from super class *Collider.* The new data field *value* of type integer describes the incrementation of health/score.

Member functions:

All functions inherited from super class *Collider*.

Classes **Shuriken, Katana** and **Kunai** inherit class *Enemy.* The methods *show()* are overridden to display different animations. Values of some data fields are changed.

Classes **Health** and **Coin** inherit class *Pickup.* The methods *show()* are overridden to display different animations. Values of some data fields are changed.

## 3.1.3 Animations

To make game look good instead of using basic canvas shapes, our team decided to use sprites[3].

Sprites are the series of frames which create an animation when played subsequently. After some trial implementations of *p5.js Sprite* and *Animation* classes, we could achieve results almost as expected. The game mechanics work so that every asset of game has one invisible collision shape (rectangular or ellipse) and one rendered animation layer.

C:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\01.pngC:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\02.pngC:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\03.pngC:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\04.pngC:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\05.pngC:\Users\lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\06.png

# Database

The database consists of one table with two columns: players’ names (type VARCHAR) and scores (type INT). It is MySQL database system and is initialized and built using UniServerZ application. As our objective is to make Javascript *score* variable ‘talk’ to PHP in order to pass the value to database, the AJAX[4] techniques are required. This way Javascript variable *score* and *name* can pass their values to database via PHP syntax. This requires XMLHttpRequest object to be declared and implemented right in the game’s main *draw() loop*.

|  |  |
| --- | --- |
| Leaderboard | |
| Name | Score |
| *Data(1…n)* | *Data(1…n)* |

# Server Side SW Design

The web service can be accessed through our student servers at *students.oamk.fi.*

Database is uploaded to *mysli.oamk.fi* student database system.

Server Side design is comparatively not complicated.

There is one and only index page subdivided into different screens. One of these screens contains the main canvas — cornerstone of “Ninja Game”. Canvas is waiting for the input commands, and sends the data to the database server when required.

If the score is among the top ten highest scores after player finish the game, the player needs to enter his or her name to the empty dialogue window.

We use AJAX to direct scores and player names from JavaScript to PHP, and then use MySQL queries to direct them to a MySQL database. After that, we select the ten highest scores to display on a separate subdivision of page.

C:\Users\lenovo\Downloads\Untitled Diagram (1).png