# ASTROMAN

# Requirements Documents

CSCI 150 Term Project – Team F – December 4, 2014


Team Members

David Hang

Edwin Perea

John Burns

Jose Garcia

# Document Contents

## Game Engine Overview and Requirements

- Introduction
- System Architecture
  - -UML Layered Architecture Diagram of Game Engine
- System Diagrams
  - - UML Game Engine and Player Use Case Diagram
  - - UML Class Diagram for handling Game Objects
  - - UML Sequence Diagram for Loading Game
- User and System Requirements

## Game Design

**An Incremental Approach**

- Develop Framework
- Drawing in SDL
- Game Functionality
- Movement and Control
- Creating and Handling Game States
- Data Management
- Creating 2D Maps
- Dealing with Collisions

**Design Issues**

- Loading Python/PySDL on Windows
  - Solved by changing to C++
- Using SDL 2.0 library
  - -Solved by having same file structure

## Game Mechanics

- General Movement

## Implementation and Testing

- Problems and issues encountered during implementation or testing
  - Merging from Visual Studio to Github
  - Key press control non-functional at one point

## REFERENCES

Mitchell, Shaun. SDL Game Development. PACKT Publishing.
http://lazyfoo.net/SDL_tutorials/

# GAME ENGINE OVERVIEW AND REQUIREMENTS

**Introduction**

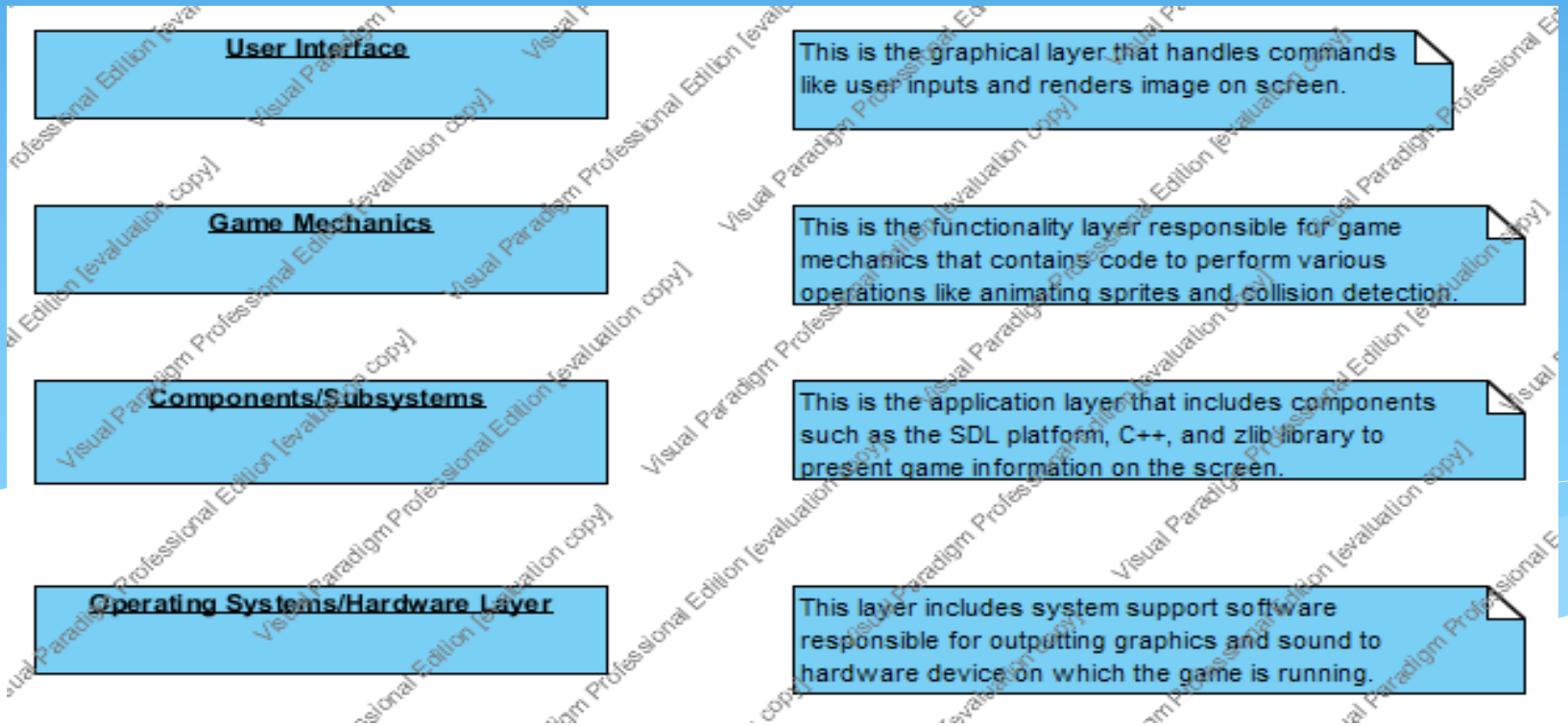
Our group will be implementing a 2D game engine. A game engine is the basic software of a computer game or video game. Most games commonly built today are with some sort of engine layer. The game engine consists of components that can be manipulated to bring a game to life such as loading, displaying, animating objects (sprites), collision detection between objects, physics, input, graphical user interfaces, and artificial intelligence. We will be using **Simple DirectMedia Layer (SDL)** with C++ to implement our game engine. SDL is a cross-platform multimedia library created by Sam Oscar Latinga. It provides low-level access to input (via mouse, keyboard, and gamepads/joystick), 3D hardware, and the 2D video frame buffer. The following software needed to implement and manage this game engine includes:

1. SDL 2.0
2. Visual C++ 2012 Express or higher
3. Tiled map editor
4. TinyXML
5. zlib library
6. Decoder64
7. Github

System Architecture

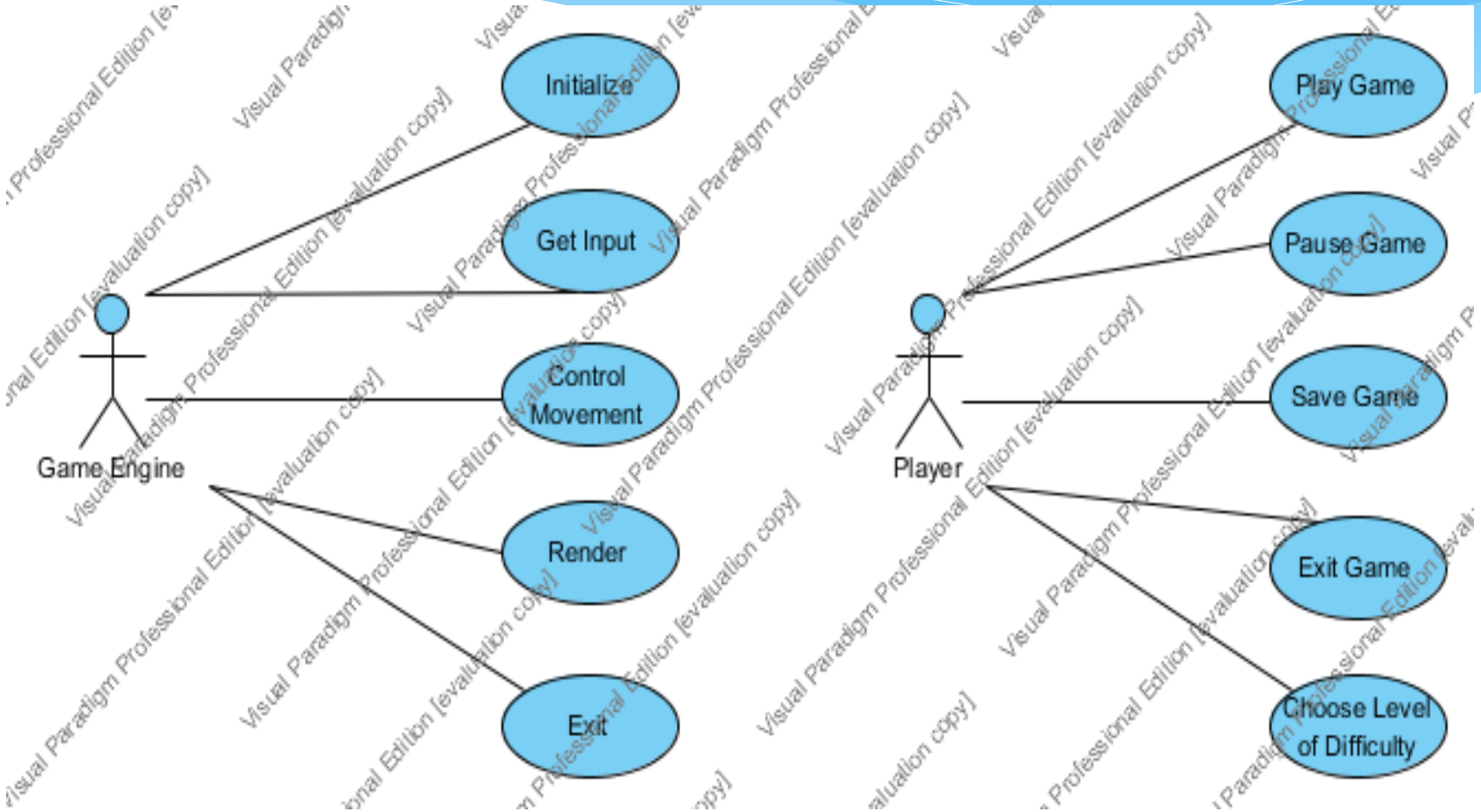
There are two major parts to game engines; the runtime component and the tools component. The runtime component is what is executed when the game is run and is responsible for displaying what the user sees as the game is played. The tools component consists of programs used to interact with the runtime components and to create content. Below is a simplify architecture diagram of the game engine which each layer only relies on the facilities and services offered by the layer immediately beneath it.

UML Layered Architecture Diagram of Game Engine

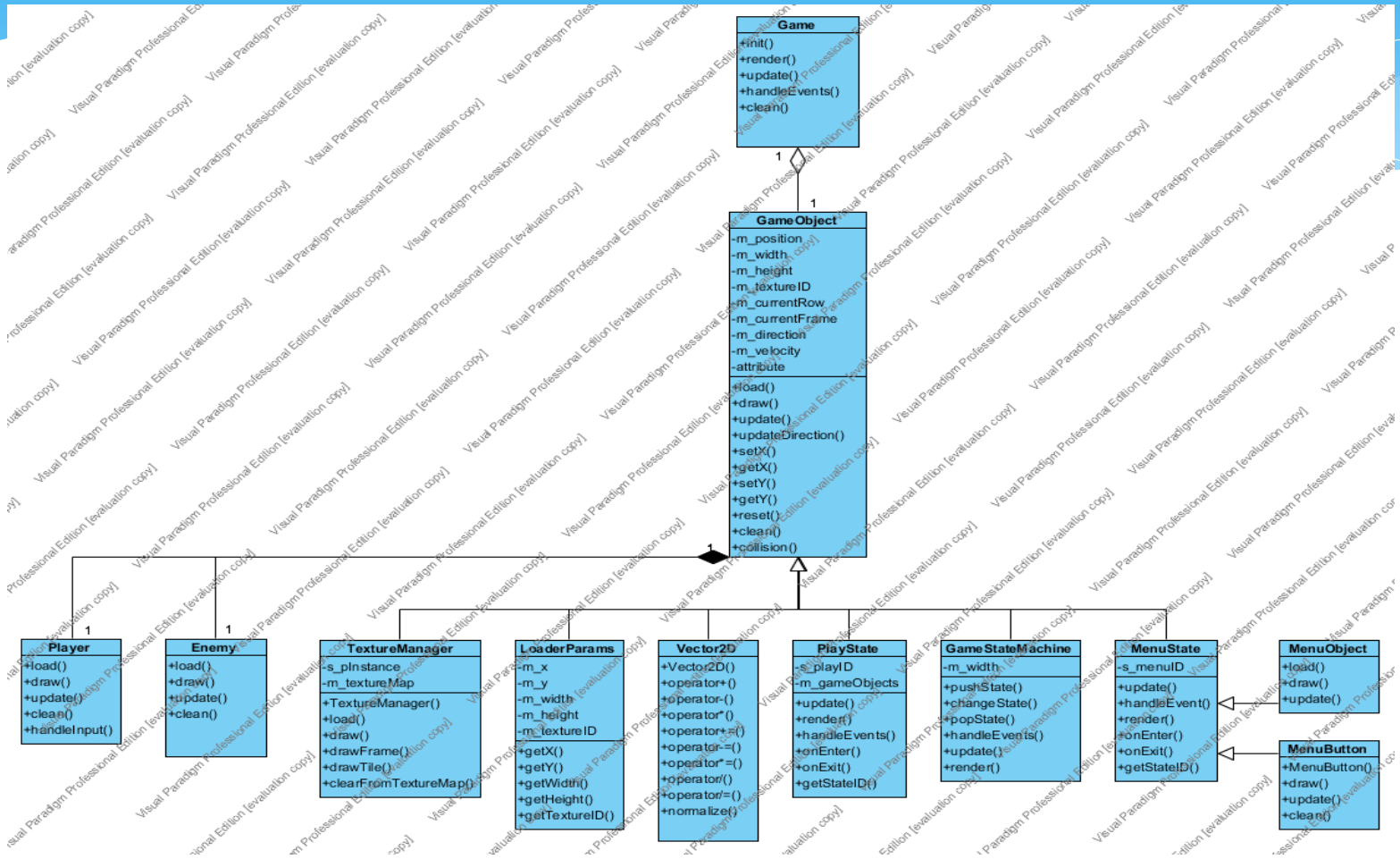


# SYSTEM DIAGRAMS

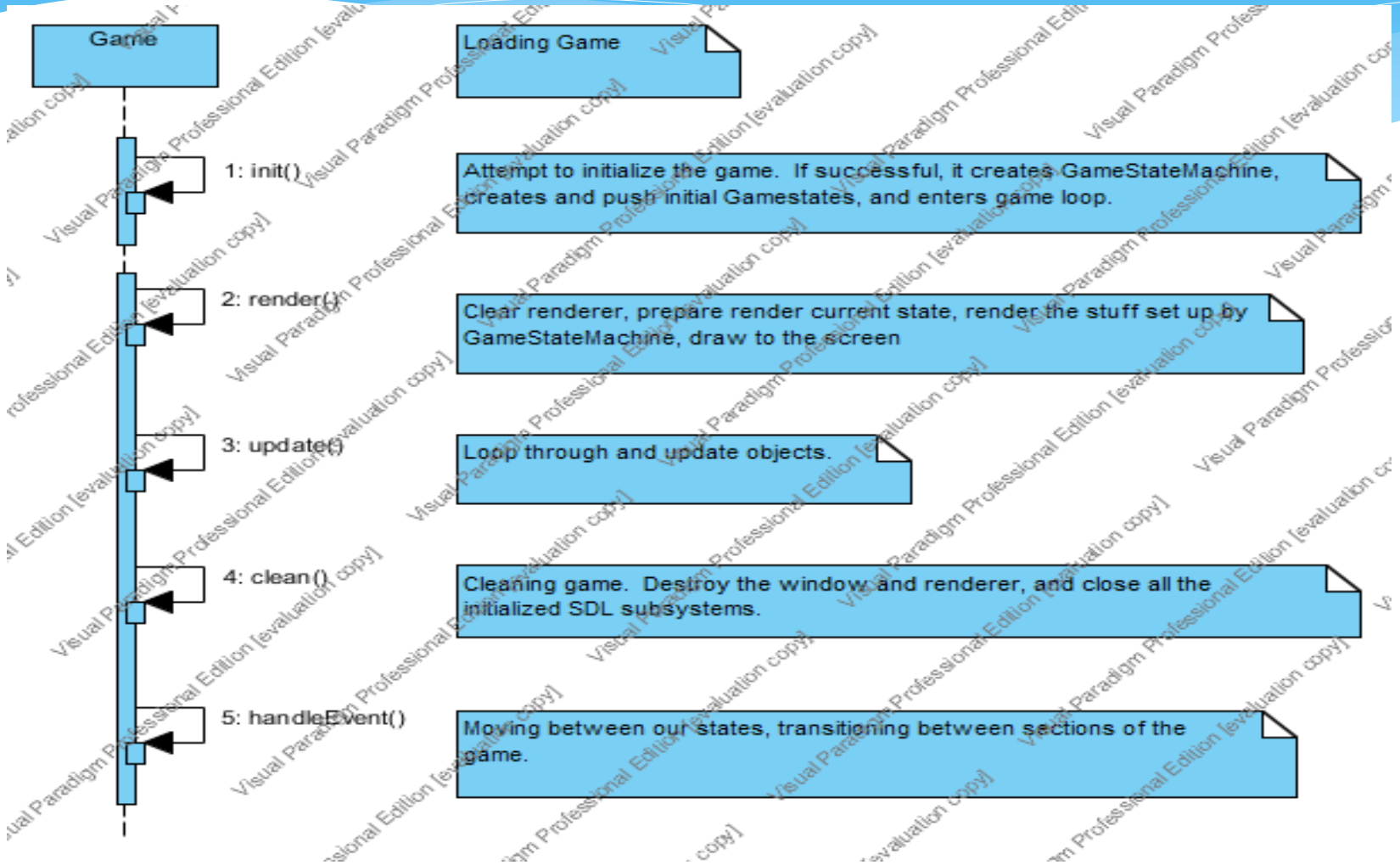
## UML Game Engine and Player Use Case Diagram



# UML Class Diagram for handling Game Objects



# UML Sequence Diagram for Loading Game



# User and System Requirements

Functional Requirement
- Playability
    - The game should start or stop when user chooses to play or exit.
    - The keyboard buttons should correspond to object movements according to user inputs.

Non-functional Requirement
- Expandability
    - Adding or changing modules and features should be made easy.
- Scalability
    - The system should be able to scale up if multiple players, objects, functions, features etc. are added.
- Portability
    - The system has to be able to run on a low-end Linux or Windows system.
- Functional system
    - It is more important to get a working, perhaps incomplete system than an unstable system packed with fancy features.

# GAME DESIGN

An Incremental Approach

1. Develop Framework

Develop the skeleton of our framework

- Building and setting up SDL in Visual C++
- Create game window with SDL
- Implement game classes

2. Drawing in SDL

Loading and displaying textures

- Get images
- Animate sprites
- Render images onto screen

3. Game Functionality

Implement game objects and interactions

- Create game objects
- Implement interaction between objects

4. Movement and Control

Setting up movement system and handling user input

- Design Cartesian coordinate system
- Implement 2D vectors
- Implement keyboard input

5. Creating and Handling Game States
Transitioning between sections of the game
- Implement finite state machine
- Implement menu state
- Implement play/pause/quit state

6. Data Management
Loading data from external files
- Load XML files using the TinyXML library
- Load other states from an XML file

7. Dealing with Collision
Dealing with object collisions
- Implement collision checking

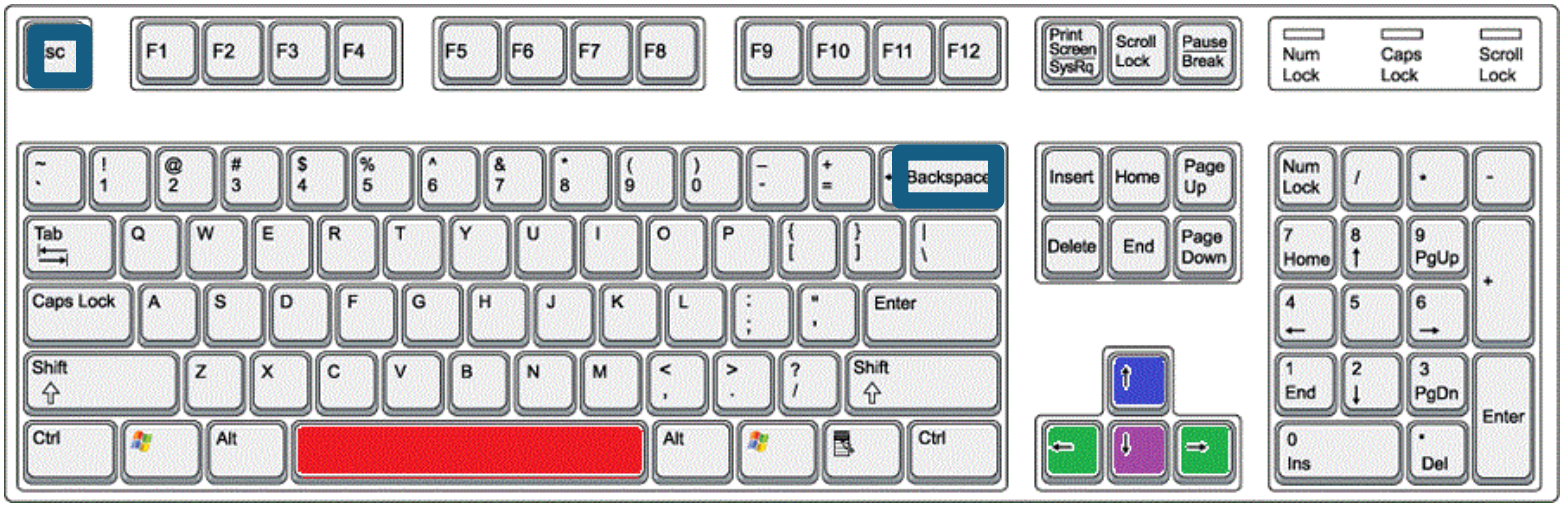
Design Issues
- Loading Python/PySDL on Windows
  - Solved by changing to C++
- Using SDL 2.0 library
  - Solved by having same file structure

# GAME MECHANICS

General Movement

The player chooses to either move or jump within the game. To move, the player has to either hit the left or right arrow (<- or ->) key on the keyboard, which moves the character in the left or right direction respectively. To jump, the player has to hit the up arrow on the keyboard.

Keyboard:




| | |
|---|---|
| SPACE BAR | Selecting options |
| LEFT/RIGHT ARROWS | Move left and right respectively |
| UP ARROW | To Jump/Selecting Option in Main Menu |
| DOWN ARROW | Selecting Option in Main Menu |

# Implementation and Testing

Test Cases

- Using frames per second (fps) to measure performance by having multiple objects

Problems and issues encountered during implementation or testing

- Merging from Visual Studio to Github
- Key press control non-functional at one point