OOAD Project Part 2

Team Members: Josh Weaver, Stephen Ham

Title: Ninja Runner

Project Summary: Our project will be a modification of an open source 2D side-scrolling, single player, platform game (Blob Wars: Metal Blob Solid) that supports a character under the player's control and a variety of enemies.

Project Requirements:
1. Player movement (primarily horizontal, but also vertical via jumping). *Constraint*: limited movement directions (left & right), and primarily via the keyboard arrow keys.
2. Real-time feedback. The game should run interactively and immediately respond to user input.
3. Attacks/collision detection.
4. Enemy AI. *Constraint*: not too advanced, just follows the player.
5. Animated sprites. *Constraint*: not all sprites will be animated due to lack of experience with graphics animation.
6. Progress indicators: showing the player the location of the controllable avatar including directions on where to go next.
7. Main menu including:
7a. Save/load
7b. Game options

8. Gameplay Goal: The goal of the game is to defeat or avoid enemy characters while rescuing other non-player characters (NPCs) that may be located throughout the game environment, and to get to the exit point.

9. Gameplay Initialization: When first loaded, the game should display an "intro" screen informing the player of a brief background story. Pressing a key should load the main menu, on which a New Game option will show slots for saved games and allow the user to select one. Selecting one will then show an option to set difficulty level, which will then show the game's "world map" where a number of locations will be selectable via a mouse click. The user must select one and then finally click to start a new game in order to get to the actual gameplay interface.

10. In-game progress/status: The game should tally a running score based on how many enemies have been defeated and how many NPCs have been rescued, including a bonus for completing a "level". The game should also be able to be paused.

11. Out-game progress: The game should allow the user to save his/her progress into one of five slots for saved games, which will be saved as data files on the computer's hard drive in the game directory.

12. Game code written in C++ and should be implemented in an object-oriented way.

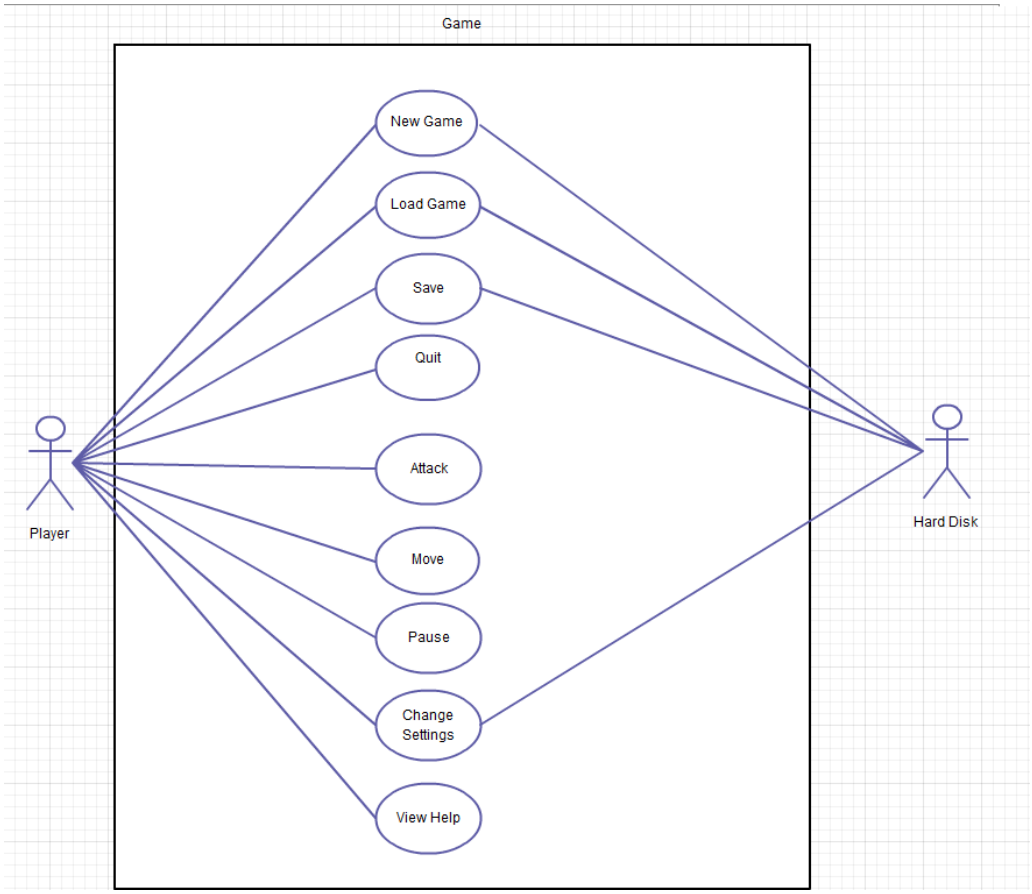
13. System Requirements/Implementation
Platform: PC, Windows XP/Vista/7 32-bit
Control: keyboard and mouse
Number of players: 1, on a single system (not multiplayer on a single system or over a network)
Setup installer: Will not be implemented
GUI-based: runs in a window, not full-screen

Users and Tasks:
Actors: Player, Hard Disk

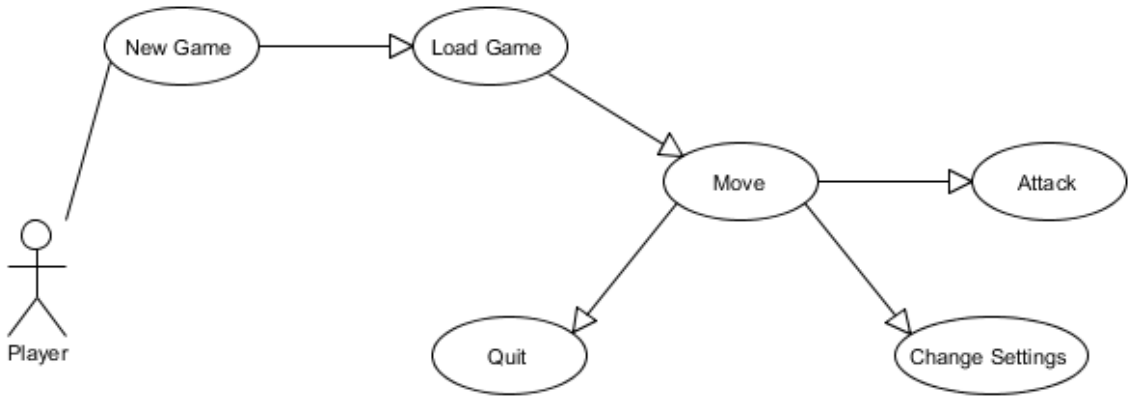
Tasks: Save
New Game
Load
Quit
Move - Left, Right, Jump
Attack
Pause
Change Settings
View Help

The system will support the following use cases as described:

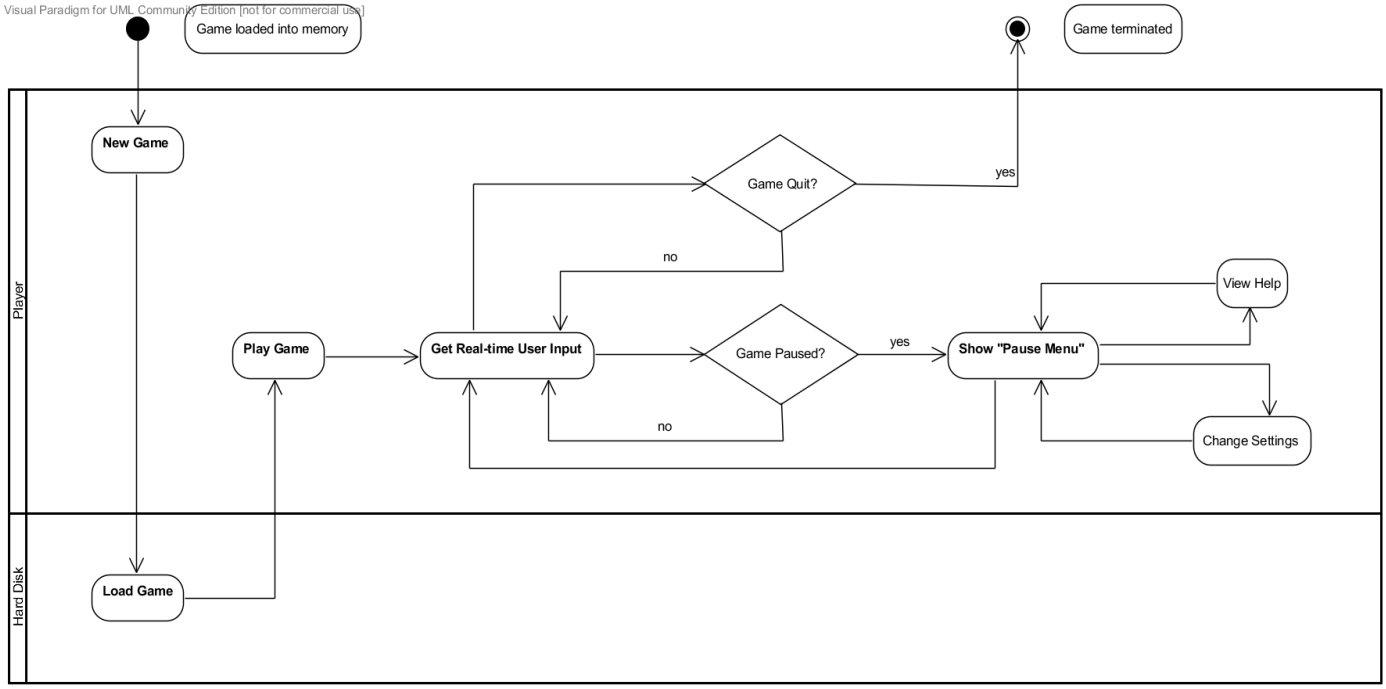
- *New Game*: changing the screen to a "Load" screen. No problems can occur since this would be just a change in screen output (unless the GPU fails).
- *Load Game*: showing the available saved-game slots and allowing the user to select one via the keyboard up & down arrow keys. Attempts to use any other key will simply be ignored by the system.
- *Quit*: allowing the user to quit the current game, or even close the window, by pressing a certain key. No problems can occur.
- *Attack*: showing an animation and effect to indicate that the player has “attacked” and caused an effect on an enemy character.
- *Move*: updating the player's avatar on screen depending on which key is pressed for the desired action: left arrow key to go left, right arrow key to go right, up arrow key for up (if in water) or jump, space key for attack. Since only these keys are defined to be accepted as user input, no problems can occur if the user presses another key.
- *Change Settings*: the player must first pause the game in order to access this option, and the game will superimpose this option over the game while paused.
- *View Help*: implemented similarly as Change Settings.



Example Use Case:

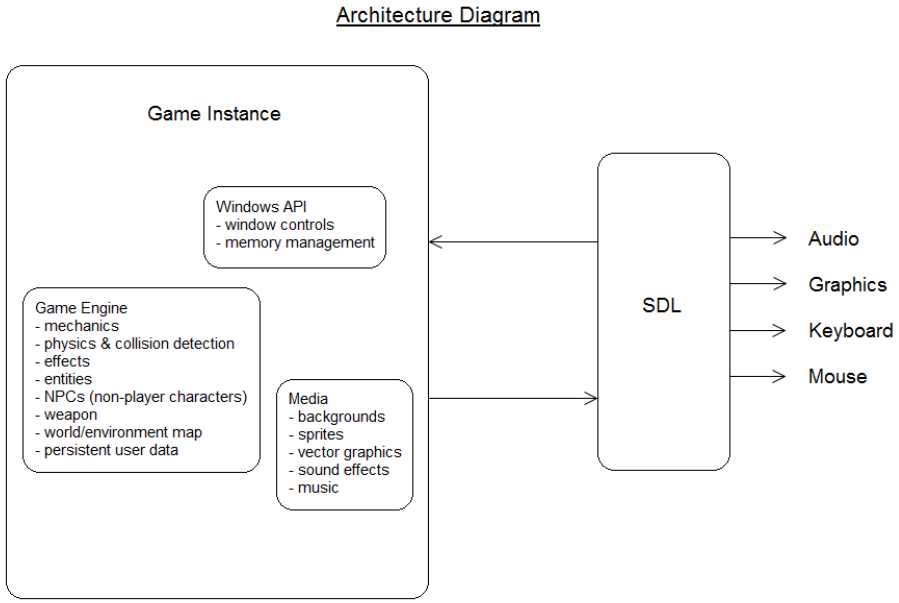


Activity Diagram:



The most complex use case will involve the following: New Game, Load Game, Move & Attack (in a series of commands that comprise user real-time input), Pause (in a sequence, so as to un-pause the game as well), Pause & Change Settings, Pause and View Help, and finally Quit.

Architecture Diagram:



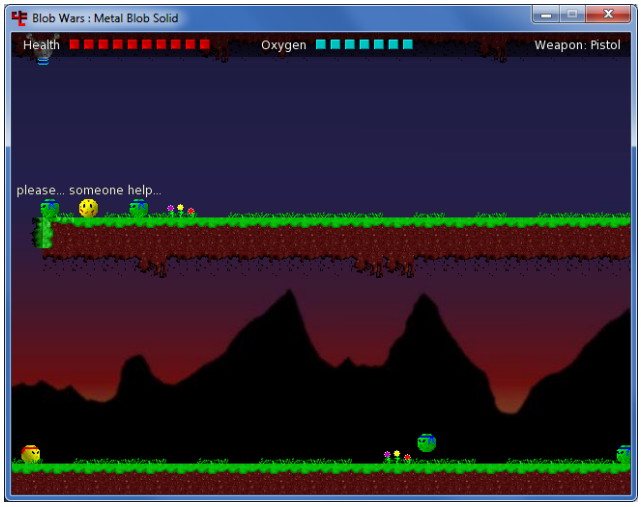
Data Storage: The data storage used in our game is going to a series of files that the objects will be written to. These files will be saved to the device on which the game is installed.

UI Mockups: Since our project consists of modifications to an open-source game, some actual screenshots from the game are provided for reference.

We will not modify the menu system (except for the title, naturally) or the Options screen. For the Gameplay screen, we plan to keep the basic Health/Oxygen meters and current weapon, along with the basic keyboard controls, but will modify entity mechanics along with other gameplay elements as time allows and possibly add new entities.

Menu System

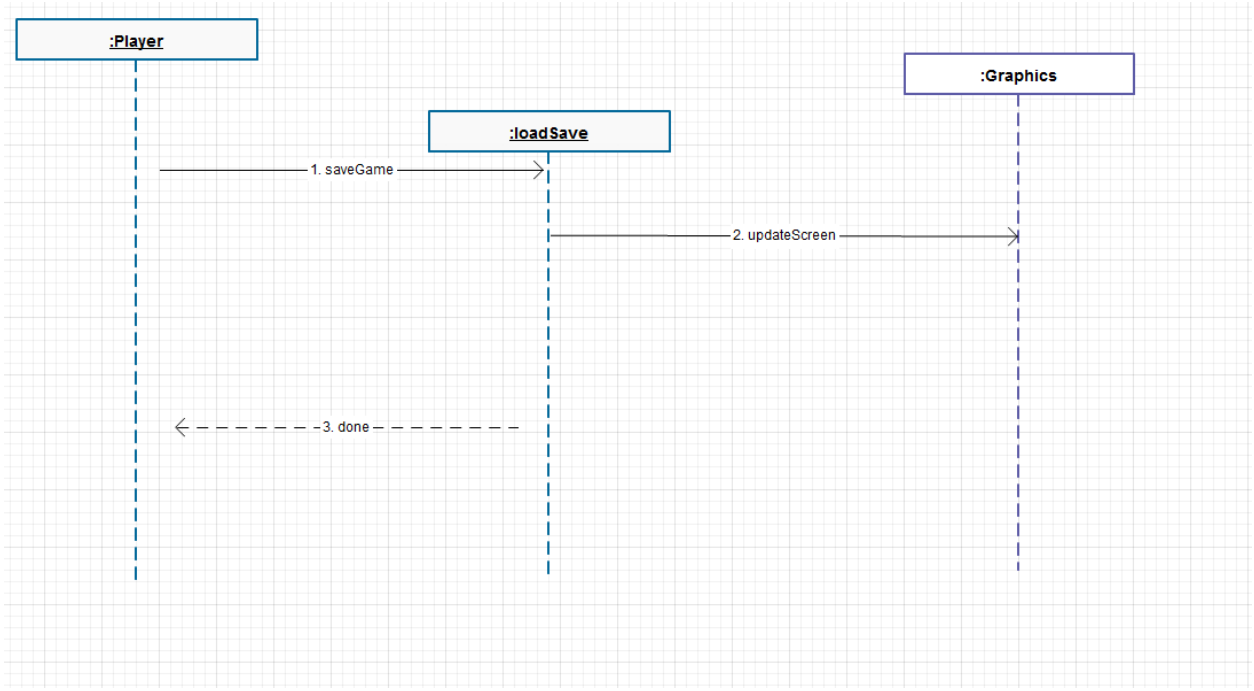
Options Selection



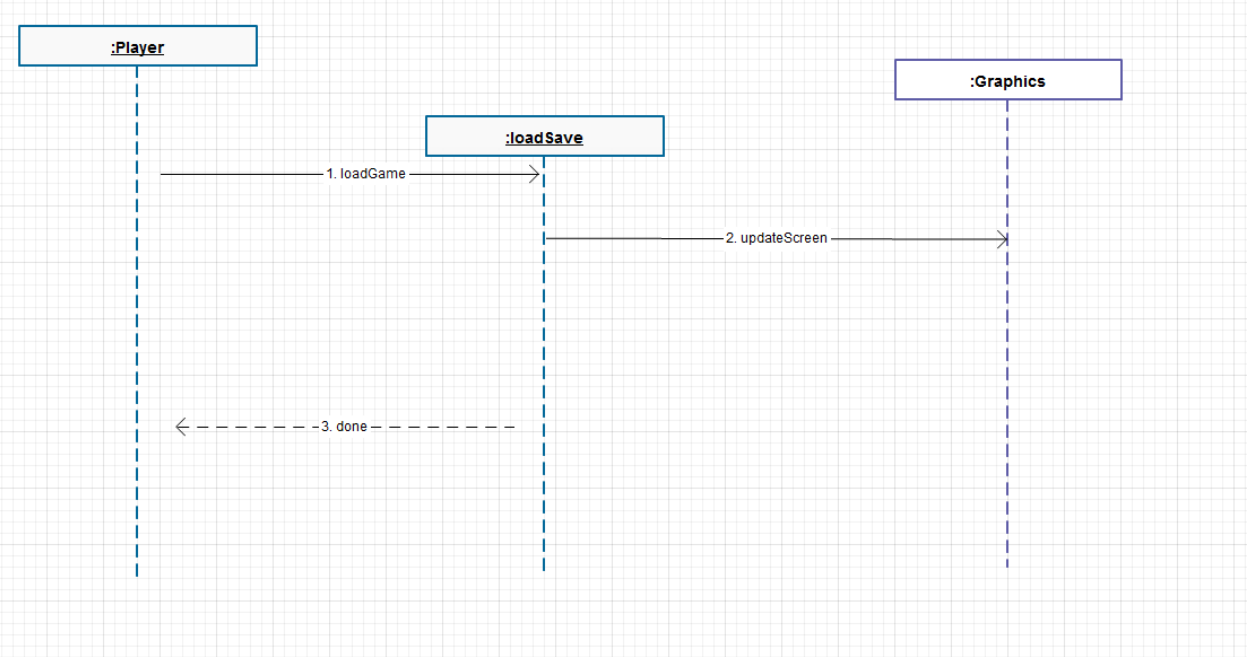
Gameplay Screen

User Interactions:

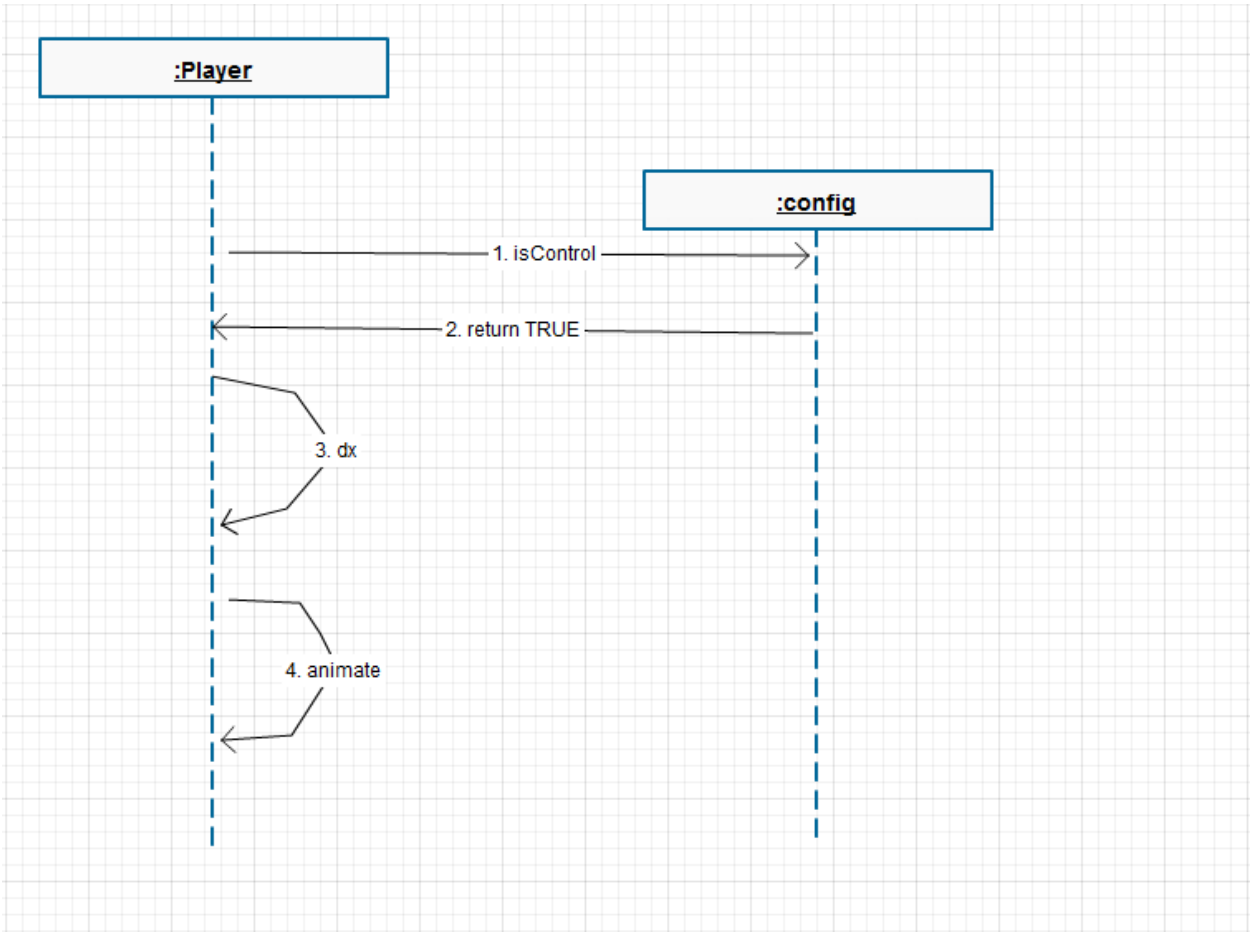
1. Save - The player will ask for the game to be saved via the saveGame() method of the loadSave Object. The loadSave object will update the graphics accordingly, write the game object to a file, then return.



2. Load- The player will ask for the game to be loaded via the loadGame() method of the loadSave Object. The loadSave object will update the graphics accordingly, read the game object from a file, then return.



3. Move - The player will ask the config object if a movement button is pressed, then the player object will move the itself by updating the dx attribute. Then the player will animate itself via the animate function.



Class Diagram: