Knight's Guard

High Level Design

COP4331C, Fall, 2014

Team Name: Group 1

Team Members:

- Megan Postava
- Katie Jurek
- David Moore
- Miguel Corona
- William Adkins
- Jonathan Bennett

Modification history:

Version	Date	Who	Comment
v0.0	09/30/14	Jonathan Bennett	Imported template, filled out Team Members and header
v0.1	10/10/14	Jonathan Bennett	Started filling out the basics for each section
v0.2	10/11/14	Jonathan Bennett	Filled out more of both sections
v0.3	10/11/14	Miguel Corona	Filled out more of both sections
v0.4	10/16/14	Jonathan Bennett	Completed "High-level Architecture"
v1.0	10/18/14	Jonathan Bennett	Finalized both sections and document.

Contents of this Document

High-level Architecture

The major components of the video game are the desktop/mobile client and the web server which is only used for the optional online scoreboard. This architecture is similar to the standard Client-Server setup, except that the server is not required in order for the client to operate.

The game will have the ability to sync the scoreboard with a web server that contains the scores of all players. When the sync function is initiated, the game will upload the local player's scores to the web server and download the scores of other players from the web server too. The player may choose to play the game and not share the score, so the web server is not needed in order to play and enjoy the game.

For the desktop/mobile client, there are 4 interfaces:

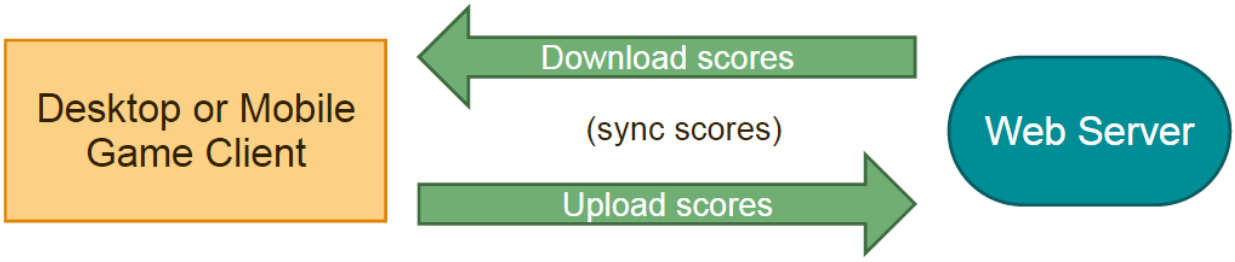
1. The title screen
2. The game screen (this is the primary interface)
3. The game stats screen
4. The scoreboard screen

The title screen is shown when the user loads the game. It shows the game logo and allows the user to click a "Begin" button when they are ready to start playing the game.

The game screen is the primary interface of the application, because this is where the player will spend the majority of their time. The game screen consists of the level map (UCF main campus), and placed on the map are the player's towers, the enemies, the player's current score and currency, and all other information and artwork related to the gameplay.

The game stats screen is shown between every new level of gameplay. This screen gives stats to the player, such as how many enemies were defeated, how many shots were fired by the towers, and how much currency the player earned in that round. It also allows the player to take a short break from the game, if needed. When the player is ready to start the next round, they can click the "Begin" button.

Finally, the scoreboard screen is used to show the player their own scores from playing the game along with the high scores of other players. These scores are kept on an online secure server. The player can choose to click a "Sync" button, which will upload their scores to the server and download the latest high scores of other players into their local desktop/mobile client. This relationship is shown in the diagram below.



Design Issues

Each section below will detail the high-level concerns and solutions of each major section that needs to be covered in order to have a successful project.

- **Reusability**
  - The design of this application will allow future reusability. The functional requirements will be designed with a specific purpose and can easily be used or slightly modified for reuse.
  - The game itself can easily make reuse of some components. For example, there are multiple types of towers and enemies. Each of these can be created by having a reusable foundational tower or enemy, and then the differences for each can be added to this foundation to create the new types.
- **Maintainability**
  - The program should be easily updated and minor issues resolved in a timely manner. The code is organized and separated by different types of objects, such as the level, the enemies, the towers, and other types of objects. By having them separated, it simplifies the process of making changes and fixing bugs.
  - Adequate code commenting is being used to identify specific functionality for each object type.
  - Easily identifiable naming convention for easy readability. For example, the variable names each begin with a character to identify its type, such as “o_enemy” which describes the variable is the Object of an Enemy.
- **Testability**
  - One possible risk is that we cannot possibly test every single computer setup and mobile device, but by adhering to standards and testing several of the most popular ones, we can be fairly certain that it will work well on others.
  - We will be testing on a variety of the most popular setups that we have available as well.
  - Each of the game objects are well separated from each other in the code base, and can be tested independently to help ensure they are functioning properly.
- **Performance**
  - The target frame-rate for the video game is 60 frames per second, which ensures a smooth gameplay experience with the game responding instantly to clicks and button pushes. The graphical requirements for the game will not be demanding for most current computers, so the target frame-rate should be easily achieved.
  - For syncing with the online scoreboard, the application should send and receive the scores from the web server within 5 seconds maximum while using a typical broadband connection.
- **Portability**
  - Portability has been achieved by our use of the GameMaker engine, which natively supports multiple types of operating systems, including the Windows and Android systems that we will use for this project.

- **Safety**
    - This video game is primarily used on the local device, so safety is not a major concern as it is not interacting with other game clients.
    - The only major safety concern is between the interaction of the client with the web server for the scoreboard sync feature. The system needs to only accept a simple and secure data transfer of scores to ensure there is no possibility of transferring harmful data.
- **Prototypes**
    - Enemy pathing prototype:
        - Make sure the pathing is following the correct paths. At “forks”, the random generation should send enemies down a random path.
    - Scoreboard prototype:
        - Make sure the scoreboard is keeping scores according to set specifications.
    - Base health bar:
        - Make sure the base (i.e. the Student Union on the level map) is capable of being damaged and the resulting damage is shown through “health bar” above the base.
    - Basic game prototype:
        - Make sure the basic mechanics are present and working :
            - Enemies spawning, with correct pathing calculations
            - Towers firing, with correct enemy targeting
- **Technical Difficulties**
    - The major technical difficulties of the project include:
        - Having enough time for all the artwork of the level maps, towers and enemies.
        - The technical details of creating the online scoreboard system and its security.
        - Having the difficulty of the game be consistent level to level, making sure it is both fun and challenging.
        - Enemy pathing; creating optional pathing based on changing dynamics of gameplay and the player’s choices.
- **Architecture**
    - The architecture is similar to Client-Server, with the majority of the work being done on the client side. The server is an optional piece that the user is not required to use, because the server will only be used to track the online scoreboard which is an optional part of the gameplay.
- **Technical Risks**
    - General:
        - There is a risk the project team’s availability may be unexpectedly limited by illness, schedule conflict, poor time management, emergencies, and other unforeseen issues.
    - Project Specific:
        - There is a risk that a software bug may be introduced into the game engine, out of the project’s team control, which affects the project.
        - There is a risk the game engine will not work as advertised on multiple platforms.
        - There is a risk the online scoreboard service may have reliability problems.