Knight's Guard

Detailed 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	10/16/14	Miguel Corona	Worked on Trace of Requirements
v0.2	10/19/14	Katie Jurek	Began filling out Implementation Locations for Trace of Requirements
v0.3	10/20/14	Miguel Corona, Jonathan Bennett	Design Issues section completed
v1.0	10/22/14	Jonathan Bennett	Finished the Trace of Requirements, diagrams and document

---

Contents of this Document

Design Issues

This section provides details on the design decisions of the game and architecture. It explains the reasoning for the decision and associated risks (if any).

- **Reusability**
  - By designing the functional requirements in a more generic fashion, they become more reusable. One example of this is the creation of a tower. By making the tower creation function more specific, multiple tower types can be created with the same tower creation function; with tower specific attributes added later.
- **Maintainability**
  - Having objects allows for easier error detection. Once errors occur, they are more easily identified and can be corrected by simply viewing the objects functionality. Objects also make it easier to update the existing code. Each object can be changed and manipulated without changing any code for the other objects.
  - Leaving consistent comments throughout the code makes the system much easier to read and maintain. By labeling what each object/function does, the overall maintenance becomes quicker and concise.
  - Easily identifiable naming makes the code easier to read as well. Being able to glance at a piece of code and know what it is doing, or at least being able to tell what variables it uses, makes editing that code easier.
- **Testability**
  - Following standard testing methods will save time and trouble. Since it is impossible to test every setup that this application can be used on, by making the application work on the most popular, we can assume that the majority of the users will have a good experience.
  - Having the components separated into distinct objects makes it easier to test specific functionality. If there is a known problem with one component of the system, instead of testing multiple pieces of the system, we can narrow down the issue to a specific object.
- **Performance**
  - The game market today is geared towards 60 frames per second(fps). This frame rate makes games appear seamless and more responsive to the users interactions.
  - The syncing process between the client and the server should be quick. This process should be quick so that the user is quickly notified of their standings on the leaderboard. This notification speed could be the difference between the user playing again or not.
- **Portability**
  - The decision to use the GameMaker engine was made in part by its support of multiple operating systems. Since the team hopes to make a game playable on both desktop and smartphone, having an engine that supports these features natively is a huge benefit.
  - There is a risk that the game will not function as well on the smartphone as it does on a desktop, due to the power available in a typical desktop computer and also possible incompatibilities in the game engine. However, research and testing gives the team confidence that the engine is capable of producing a smooth output for smartphones as well as desktop operating systems.
  - There is an additional risk of running out of time, because there will need to be adjustments made to the smartphone version of the game. The team is focusing on completing the desktop version first, and depending on the remaining time available,

the smartphone version may not be ready for release, or not play as well as its desktop counterpart.

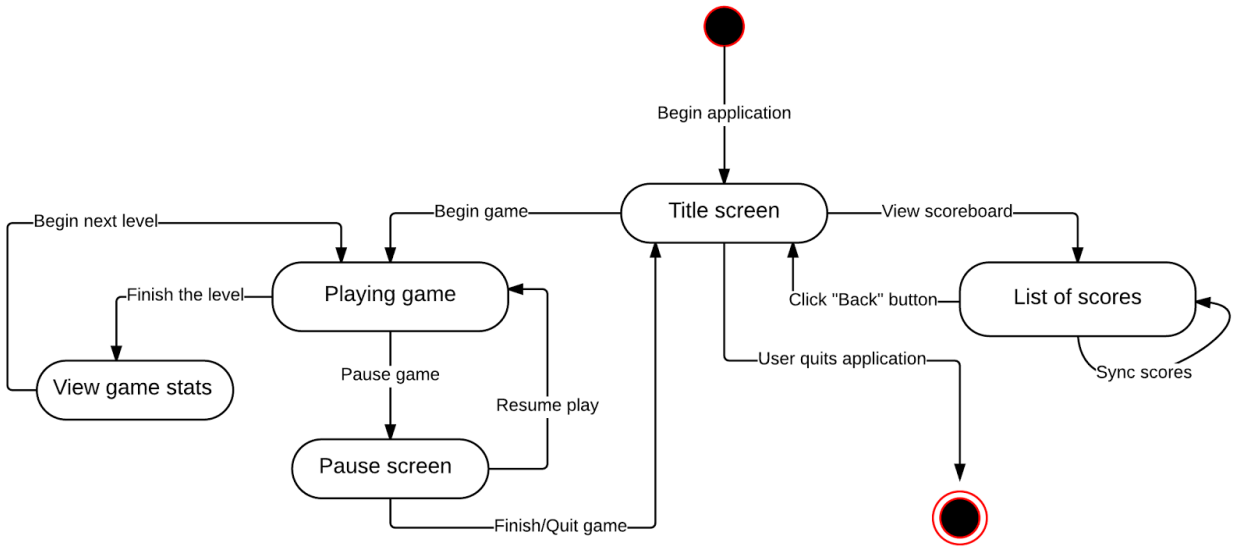
- **Safety**
    - Since this video game is primarily used on the local device, the risks for safety are quite limited (compared to software such as a banking application). The only risk involved is the networking between the local client and the web server that is used to track the scores of all players, i.e. the online scoreboard.
    - There exists a risk for this scoreboard to be manipulated, such as by a player who may try to sniff the network data in order to learn how to send fake high scores to the web server. The development team is taking steps to prevent this issue from occurring, but the ultimate risk is minimal because the online scoreboard is not used for anything except keeping track of text scores, which are optional and play no part in keeping the game operational on the local devices of all players.
- **Prototypes**
    - Enemy pathing prototype:
        - This prototype was useful so that the spawned enemies would not overcrowd a single path. It also adds a random element to the game because enemies have the choice of multiple paths to take, instead of taking the same predictable path each time. This will make the game more challenging and fun.
    - Scoreboard prototype:
        - The scoreboard is a classic video game pattern and adds an extra element of fun and competitiveness to the game. It gives the player a reason to try again and improve their score. It also allows multiple players to compare their abilities against one another by comparing their scores.
        - The player is able to gain score by efficient use of their towers and killing enemies before they can attack the player's base. This prototype enabled the team to start implementing a basic scoreboard into the game, which increased by 100 points for each enemy kill.
    - Base health bar:
        - The main objective of the game is to protect the Student Union base from the attacks of the enemies. The base is capable of receiving multiple attacks over time, and the game needs a way to show the player how many attacks it can withstand. This is achieved by use of a health bar, which is conventional in video games. As the base gets attacked, the health bar will continue to decrease. The player may be able to restore health to the base with special powerups in the game.
    - Basic game prototype:
        - This was the first prototype made available to the client.
        - The prototype was focused on getting the basic mechanics of the game working. The prototype featured a basic 1-color map to play on, one location where enemies spawn, one location for the enemies to move to (using A* path-finding) and the ability to place towers anywhere on the map that would automatically shoot at the enemy.
        - The team chose to focus on the basic mechanics because they serve as the foundation of the entire game. In order to develop more complicated towers and enemies, we are using a basic version for each of them to start with.
        - The prototype was shared with the client for additional feedback and to show progress. This allowed the team to gather feedback on the game so far and help

reduce risk of misunderstanding or wasted time focusing on features that shouldn't be in the game.
        - The next prototype will build on these basic mechanics and introduce more complexity, such as additional details added to the user interface and multiple enemy paths.
- **Technical Difficulties**
    - Since the majority of this project relies on the implementation and design of the code that allows the system to function, the last aspect we need to finish is its artwork appearance. The artwork needs to fit the overall theme of the game or the game can feel cheap.
    - Making the enemies follow predefined paths and allowing these paths to change with a random generated value can be tricky. The enemy pathing will not only be effected by "forks", lane branching, but also by the users placed towers. In the end, designing and implementing these path choices will make the game more interesting and unpredictable.
    - Game difficulty is a huge part of what makes a game successful. If the game is easy to playthrough and presents no challenges, the user may become bored. If the game is very difficult the user can get frustrated and give up. The game needs to be balanced. This can be achieved by use of playtesting and being mindful of successful game design patterns.
- **Architecture**
    - The Client-Server architecture was chosen because it best models how the system will operate. The majority of the code and gameplay is on the local device, whether it's the desktop or the smartphone. The client will function properly, even if the web server is not available.
    - The only feature the web server is used for is the online scoreboard system, where players can sync their scores with the web server to upload their own high score and download the high scores of other players for comparison. This feature is optional for the player and does not affect the gameplay.
    - There are two risks with the server:
        - There is a possibility that the online scoreboard feature will not be completed on time, if the development team runs into a time crunch or problem with the functionality. It is not considered to be as high of a priority as other game features.
        - There is a possibility that the web service may experience downtime, or the player will not have a network connection available. The game client will need to account for these possibilities and be able to handle them gracefully.
- **Technical Risks**
    - General:
        - The development team is composed of Computer Science students, several of whom are also working part-time/full-time jobs while attending school. There is a risk that work on the project may stall or slow if the schedules of team members become too hectic.
        - There is also a risk that team members may become ill and unable to contribute to the project. Since the work is being equally divided, the other team members will help pick up the slack, if needed.

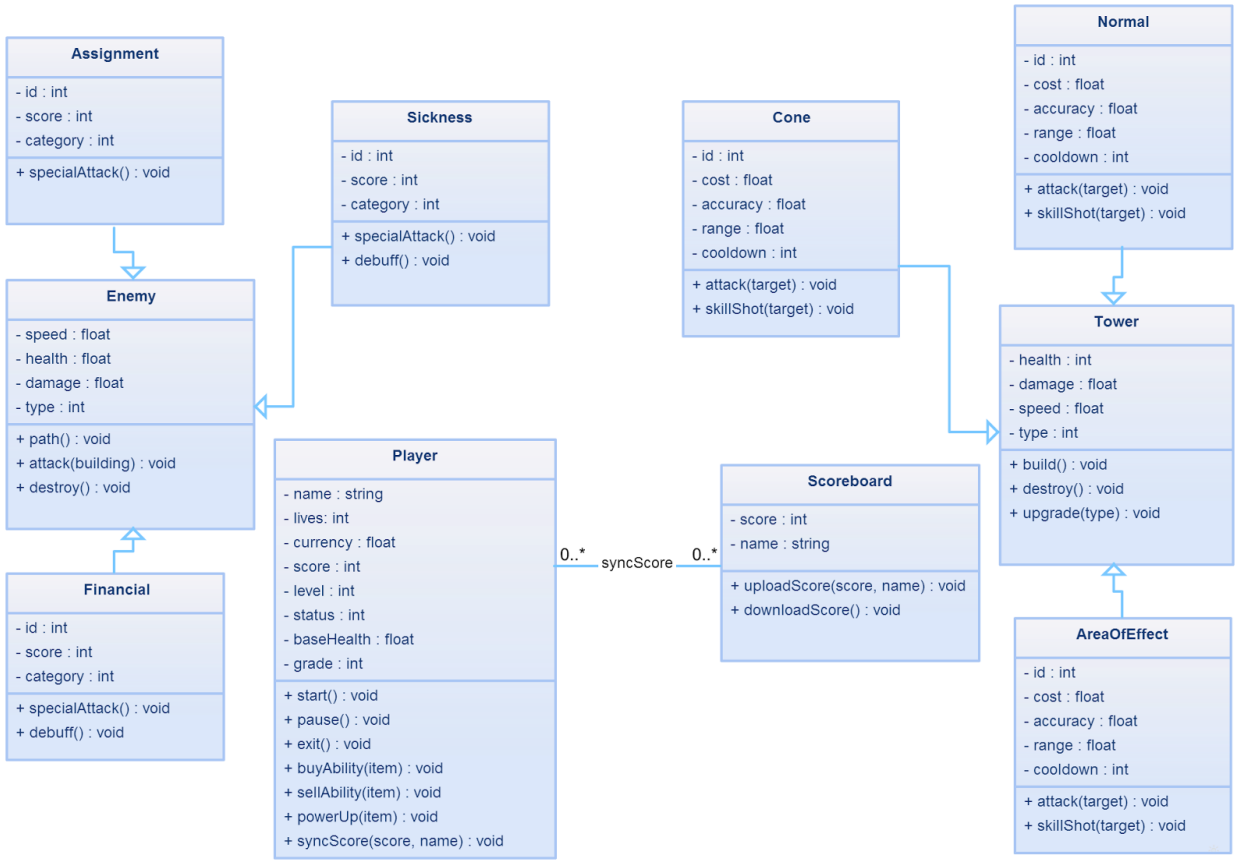
- Project Specific:
  - The project is dependant on the GameMaker engine. If the GameMaker team introduces a game-breaking bug into their engine, this could affect our game.
  - The project will work on the Microsoft Windows platform at a minimum, but the goal is to support the Android platform as well. The GameMaker engine supports multiple engines, including other platforms such as Mac OS. Our team is relying on this feature to work correctly, but if it doesn't then there is a risk that the Android platform's performance may suffer or not be released in time.
  - There is a risk that our online scoreboard service may become unavailable or change its API, which would require us to update our application in order to keep it functional. This risk depends on if an external or internal service is used. The advantage of an external system, such as Google Play Services, is that it is well-tested and potentially quicker to begin using. The disadvantage is that it has the possibility of changing without the development team's awareness, which is not something that would occur if using a custom-made solution.

Detailed Design Information

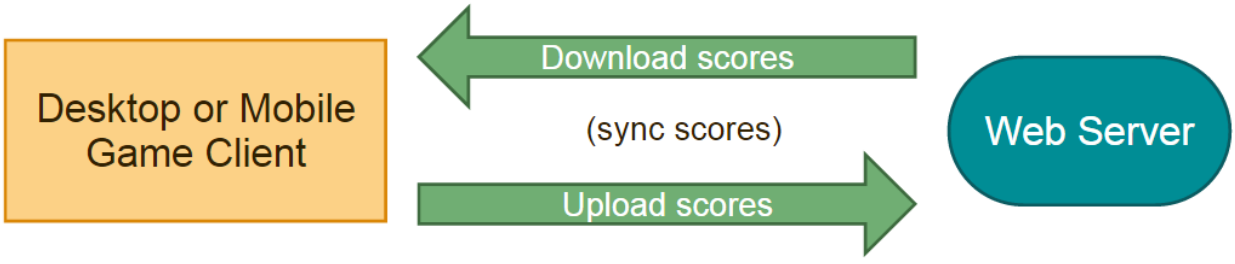
1. State diagram: Using the application



2. Class diagram



3. High-level overview of system architecture



Trace of Requirements to Design

Requirement From SRS	Implementation Location
1) The system shall allow the user to start the software.	If the user double clicks on the Knight's Guard .exe, the software will start.
2) The system shall allow the user to start the game.	If the user clicks on the start button on the screen once the software is loaded, the game will start.
3) The system shall allow the user to quit the software.	The exit button (the X) will be located to the top-right of the game window. Also, pressing Escape twice in a row will exit the game.
4) The system shall allow the spawning of enemy units.	The spawning of enemy units is handled in o_control's Alarm 1 event, which spawns enemies at various spawn points on the screen.
5) The system shall allow the user to place towers.	If the user has an available tower to place, they may place it by clicking on "mount points", or points where they are allowed to have towers.
6) The system shall allow the user to upgrade towers.	If the player clicks directly on a placed tower, that tower's upgrade options will be displayed.
7) The system shall allow the user's base to take damage/be attacked.	If the enemies are close enough to the base to attack it, they will begin doing damage to it until they are destroyed by the user's towers.
8) The system shall allow the user to pause the game.	If the user presses "P", the game will pause.
9) The system shall allow the user to restart the game.	If the user presses "R", the game will restart.
10) The system shall allow the user the option to quit the current game.	If the user presses the Escape key, they will back out to a menu. If they press it again, the game will be exited.
11) The system shall allow the user to progress through levels.	This is an automatic part of the gameplay as the user finishes a level.
12) The system shall allow the user to finish the game (Game Over).	This is an automatic part of the gameplay as the user finishes all levels.
13) The system shall allow the user to interact with the user interface.	The user will interact with the UI by clicking on buttons on the screen and pressing keys on their keyboard.

14) Development of the system shall take place in an environment capable of coordinating with a git repository.	GameMaker: Studio is compatible with using a git repository. The “GitHub for Windows” software is primarily used.
15) The game shall run on the Windows 7 operating system.	This is handled by the game engine (non-interface).
16) The game shall run on the Android 2.3+ operating system.	This is handled by the game engine (non-interface).
17) The system shall prevent the malicious change of scores.	This is handled by the game engine and the code base (non-interface).
18) The system shall provide the average player a fun experience.	This is handled by play-testing and following game design standards (non-interface).
19) The system shall be easy to learn to use and play.	This is handled by play-testing and following game design standards (non-interface).
20) The documentation shall be readable and posted online.	Documentation will be available on the Group 1 public web site (non-interface).
21) The video game shall provide a guide on how to install and play the game.	There will be a user manual included with the game, located in the game’s install directory.
22) The system shall keep track of the number of units spawned.	There will be a numEnemies variable that keeps track of how many enemies have been spawned and are currently alive on the screen.
23) The system shall keep track of the damage done to the base.	The base will have a health variable that stores its current health.
24) The system shall keep track of the number of levels completed successfully.	There will be a numLevels variable that keeps track of how many levels have been completed. This number may be shown on the screen to the user in-between levels.
25) The system shall be created by individuals with at least basic knowledge using GameMaker: Studio.	Non-interface requirement.
26) The system shall be created by individuals with experience coding.	Non-interface requirement.
27) The system shall prevent users from changing system settings.	Non-interface requirement. This is handled in the code base and game engine.
28) The system shall be easy to navigate and	This is handled by use of play-testing to identify

readable.	issues and following conventional design patterns.
29) The video game must play smoothly.	The code will be inspected for inefficient methods of solving problems and be emptied of them to the best of the coders' knowledge and abilities.
30) The video game must be well tested.	This is handled by extensive play-testing by the development team and volunteers.
31) The video game input should feel responsive.	This is handled by extensive play-testing by the development team and volunteers, along with following coding practices optimized for performance.
32) The project must meet the client's expectations.	The game will be fun and simple to learn.