**Game Report**

Team 7

Game: Gun the Gamblers

**Game Concept:**

Gun the Gamblers is a 2D scrolling shooting game. The graphics of this game is meant to be very basic because we are not designers. Gun the Gamblers takes the setting of a dark and grimy era, where gambling has become a disease and gamblers are trying to steal money to gamble more. And if someone’s money are gone, it just makes them all the more prone to gambling and win back that money. The player takes the role of a person chosen by the Hospital to knockout the gamblers, so that the Hospital can treat them.

**Gantt Chart:**

Gantt Chart is attached to the file.

**Levels:**

The game has 3 levels. Each level contains more than 300 entities. The first level is more open, letting the player have more movement. This lets the player have freedom and is essentially easier for the player. Level 2 has more corners, thus restricting the movement of the player. With his movements constricted, the gamblers have an easier time chasing. Level 3 is meant to be a mixture of both; half of it is open, while the other half is restricted.

**Changing of Features:**

During the course of the development, Wei Jie fell heavily behind and we had to take over his feature(gun) because he took too long to do it. Furthermore, his other features (casino, buildings) had to be shaved off, because it took too long to create.

**Problems Encountered/ Problems Solved/ Lessons Learnt:**

Most of these are answered in the TDD, as we were more focused on the technical aspects of the game than the visual aspect and the playing of it. To reiterate  
  
**CHALLENGES AND SOLUTIONS**

One of the main problems we faced almost immediately was code rigidity. Because the entire framework depended on the use of the Entity class on a vector, it meant that codes were limited to only those on the entity class, if we were to run it on other classes. Furthermore, the entire hierarchy meant that it was quite difficult passing information around from one entity to another. To overcome this, we had to either use polymorphism or make more virtual functions. We decided to come with a mixture of both.

Another problem that came was the oversight of the game itself. To be more specific, how many entities there would be in one level. For a level to even be remotely considered a level, it contained more than 300 entities, mainly consisting of useless building classes. With this amount of entities, the FPS dropped in 5 fps and below. To optimize this, a quad tree was created.

Lastly, the members in the team did not know how the entire program flowed, even after explaining it. After teaching it to them, they slowly understood and were able to use it. This was a great hamper to the project, since there was a member that implemented his own framework into the project. Massive time was lost in taking out the framework and fixing the feature. Because the members did not understand the framework, there were lots of bugs that appeared that they could not solve. The implementation of the classes into the Model and the View, as well as the debugging of most of the bugs could only be done by one person. This further hampered progress and gave a high load of work on the one person.

**LESSONS LEARNT**

The framework used for the project was very dependent on planning, and the members understanding the framework of the project and how it ran. When we started out, we had planned, but we did not go into enough detail. As such, software rigidity quickly appeared, and we had to adapt the framework to save time. Furthermore, this framework does not allow for a programmer’s own style of programming; it forces a programmer to code the way the framework dictates. For starting programmers that do not even have basics of pointers down, this is incredibly difficult. Furthermore, the planning of the project was not perfect either; we had a lot of holes in in the planning that was overlooked.

If given another chance, I would not recommend using this framework to programmers that do not have the discipline or the base knowledge of inheritance and pointers to use. This framework requires too much planning and is too rigid to be worth the tradeoff for optimization. By rigid, I mean that it is tough to modify code. Once a code is linked to the project, it becomes tedious to modify it. However, the positive side to this framework is that it is quite simple add more features and entities to the game.

It would have been much easier for the programmers to use the framework Mr.Tang gave; one using one single object and using enums to determine the object. Although the memory used for this type of framework is greater (due to the unused memory spaces), it allows for greater flexibility in code and more modifications. Furthermore, since most of the members have at least used it once, they are all familiar with how the programs flows.