**Star collector**

After studying the effect of programming languages on code clones for games, we analyze how categories impact code clones. We study five specific categories (Table 1) in this paper which are First Person Shooter Game, Card Game, 3D Game Engine, Role-Playing Game and Chess. Figure 7 summarizes the PTCLOC for chosen genres of games. As Figure 7 illustrates, basically, the PTCLOC of different categories are consistent to their languages. First Person Shooter Games – C-based games – still are the higher in all three types of clones, which have the highest PTCLOC. Moreover, among the Java-based games, Chess games keep the lowest level of code clones, from 0.91% of Type-1 Clones to 11.10% of Type-3 Clones, whereas 3D Game Engines and Role-Playing Games achieve slightly higher value (still very low) where the Card games are placed in the middle. As a result of this investigation, we answer RQ2 as follows: basically, the characteristics of code clones in different categories are consistent with the corresponding programming languages. First Person Shooter Games have the highest PTCLOC, following by Card G.

**Clone-coupling (CCP):**

CCP (Equation 6) measures to what extent the code fragments in a file or a directory have clone relationship with other code fragments in different files or directories. In CCP, ݊ denotes the number of the clone classes associated with the target file or directory in the games, and the “foreign members” refer to those members of the associated clone classes that are in different file or directory. For example, a high CCP means that the members of the associated clone classes have clone-pair relationships with many distinct clone classes scattered in the same file or directory. Unlike to CCH, a lower CCP of a game is desirable in our study as such characteristic minimizes the effort to understand subsystem functionality in dealing with the local clones.

The touch screen is used to navigate the user menu. The camera, the screen, the accelerometer and the compass will be used to support the VR aspect of the game. The phone’s orientation obtained from the sensors will be used to aim within the virtual space. And as discussed earlier, the camera deals with overlaying virtual targets on the display. Audio output will be used to provide in game feedback to the players.

**3D Models**

Because of limited time, the design and creation of 3D models are out of scope of the project. We planned to use free ready-to-use 3D models in the project. However, at the beginning this was a nightmare for us. If we did not find suitable models, making a game would be a fantasy and accordingly the project would fail. Fortunately, we found the suitable models in unexpected short time. We selected models from free models on the web sites selling models. At the end of the project, we realized that while we think finding 3D models as a nightmare, it becomes a stage that gives rapid results and makes us very happy.

**Limited Time**

The main problem for the project was limited time. Although the project is small, we think that 63 days are not enough for implementation. Due to the fact that we had no game development experience, we had to do a considerable amount of research during implementation. This did lead to slow down the progress of implementation of the game. Fortunately, we did not come up against the case of unfinished implementation work. This situation resulted in just putting increasing stress on us and acting as if we had panic attacks during the entire implementation phase. We felt the impacts of limited time during not only implementation stage but also testing stage.

**Integration of 2D and 3D**

We built our game in a 3D world and our menu system in a 2D platform. Our game and menu systems were working properly before integration. While they were working separately, everything was perfect. When they came together, the problem showed up. We found out very good sample codes for 3D game world and GUI. We reviewed all the samples we could find. However, almost all were for the sections where we have already not encounter any problem. We could not foresee how difficult to find a simple sample showing the way of the two working together.

The leading cause was that we implemented the 3D world and GUI as two separate game frameworks. Due to being two different games, XNA game engine selected one of them and run it.

We fixed the problem by referencing the game framework including menu systems to the one including core game.

The game has a design with only single-player mode. In the single-player mode one can find three sub-modes: easy, normal and hard. Depends on sub-modes, the number of enemies and pick-ups will vary. When the game becomes harder, the number of enemies will increase and the number of pick-ups will reduce. The player starts the game with two weapons and certain number of ammo. During the game, one has the ability to pick up more weapons and ammos. Along the way one face off against certain number of enemies which depends on the type of sub-mode. By default, the player has one hundred health units. The attack of enemies reduces the health primarily.

The effect of damage varies depending on the type of attack. The enemies can bite; they do not have the ability to use gun and shoot. However, the player can pick up the power-ups that increase health. The player has only one life. When his/her health reduces to zero or lower, one will die. When our main character dies, the player loses the game. If the player kills all the enemies before the health reduce to zero or lower, one wins the game.