# Design Overview for Artillery 3

Name: **Jimmy Trac**

Student ID: **101624964**

# Summary of Program

**Artillery 3** is a 2D physics-based shooter where players take turns controlling vehicles on a map of varying elevation. Players will be able to move their artillery pieces across the map and fire in long arcs towards enemy players, with the explicit goal of destroying all other players. The game will be turn-based, with players selecting a weapon (if not multiple) and select an angle to fire at. The “force” behind each shot will be a rough estimate at best and the fired projectiles of varying properties and physical interactions will both damage players’ health, armour, and stats, along with the terrain itself. Figure 1 is illustrative of the initial-initial concept for Artillery (the original version!)

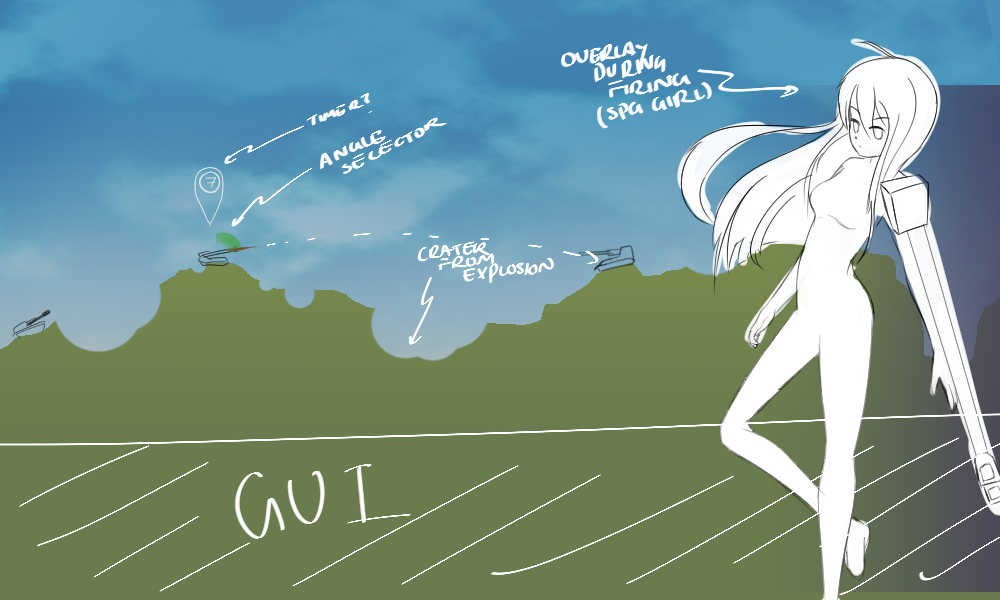


Figure 1 – Example Game Sketch

# Required Roles

Describe each of the classes and interfaces you will create using the following table (one per record).

Table : <<role name>> details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
|  | Field type, parameter and return types |  |
| Player | Class, health, character, position, selected weapon |  |
| Weapon |  |  |
| GameManager | Currently selected player, Camera, game state  “the world” | Maybe a “world of artillery” class is better for describing the game manager |
| Physics | Physics simulation, physics components | Entities that can collide have physics. If they can move then their movement is described by the engine based on the entity’s PhysicsComponent data |
| Character | Animations, Character States, Bitmap, weapon, health, armour, shield, power-ups | The “vehicle” of A3 |
| Terrain | Terrain, any buildings | Terrain factory to make the different terrains based on different algorithms |
| Camera | Tracking entity, position, speed, easing function | Camera object is used extensively in games, though I’m not sure about the specifics |
| Commands | Player to execute on, keyboard input (maybe controller input too) | Used for the command design pattern |
| Projectiles | Colour, bitmap, physics, animation, collision, damage, type of damage, armour penetration | Fired by weapons, needs a specific projectile, maybe has a manager, physics. |
| Tracers | Colour, position, fade time, fade function | Created by projectiles, pretty. |
| Effects | Animation function, effect, maybe special physics? | Columns of magic that rain down from the heavens! |
| Particles | Position, colour, fade, initial velocity, special movement (e.g. spin), transparency | Physics, like tracers but affected by gravity |
| Entities | Position, bitmap, updates, draws | A basic game object that can contain components. Describes players, characters, destructible things, projectiles, and similar things. |
| GameObjects | Not sure if this will be implemented | Base game object, both updatable and drawables inherit from here maybe. |
| EntityManager | Singleton  List of entities, add, remove, manage entities. Updates them. | Manages entities.  Contains list of entities,  Entity add, entity remove, entity simulation. |
| UI Events | Not sure | Observer events vs. C# Events. |
| GUI/Interface | Not sure | See above. No idea right now. |
| Wind | Might be a component of the Physics Engine | Should be physical property of the world maybe, maybe even a part of the physics engine |
| Parallax Environment | Might be a component of the world | Multiple terrains together, or is there a “world” that has multiple terrains? |
| Shop | List of buy-ables  Might be a part of the menu system | Game state that allows upgrades. Like menu |
| Inventory | Might be implemented later. | Maybe for special ammunition purchased from shop |
| Destructible | Health, bitmap, position, physics (for collision) |  |
| Music/Sound | Sound files, singleton. |  |

Table 2: <<enumeration name>> details

|  |  |
| --- | --- |
| Value | Notes |
| GameState |  |