

# Design Document

Anders Petersen (petersea@uci.edu)

Bob Nguyen (boobio@uci.edu)

Shaun Shue (sshue@uci.edu)

Charlie Hui (chui@uci.edu)

Hieu Le (leh@uci.edu)

May 7, 2003

# Contents

<b>1 Overview</b>	<b>2</b>
<b>2 The Rules</b>	<b>2</b>
2.1 Basics . . . . .	2
2.2 Various modes of play . . . . .	3
2.3 The tanks . . . . .	3
2.4 The Playing fields . . . . .	4
2.5 Powerups . . . . .	5
<b>3 AI for the game</b>	<b>5</b>
3.1 Very simple AI . . . . .	5
3.2 More advanced AI . . . . .	6
<b>4 Artwork and user interface</b>	<b>6</b>
4.1 Artwork . . . . .	6
4.2 User Interface . . . . .	6
<b>5 Sketches (to be finished)</b>	<b>7</b>
<b>6 Music and Sound</b>	<b>7</b>
6.1 Music . . . . .	7
6.2 Sound Effects . . . . .	7
<b>7 Unit Descriptions</b>	<b>7</b>
<b>8 Technical Specs</b>	<b>8</b>
8.1 Platform . . . . .	8
8.2 Game Engine . . . . .	8
8.2.1 Tank Behaviour . . . . .	9
8.2.2 Collision detection . . . . .	9
8.2.3 Output Interface . . . . .	9
8.2.4 Input Interface . . . . .	9
8.3 Tank Specification . . . . .	10
8.4 Tileset Specification . . . . .	11
8.5 Level Specification . . . . .	11
8.6 Graphics Engine . . . . .	12
8.6.1 System Module . . . . .	13
8.6.2 Graphics Module . . . . .	14
8.6.3 Input Module . . . . .	15
8.6.4 Sound Module . . . . .	17
8.6.5 Network Module . . . . .	19
8.7 Version Control . . . . .	22
<b>9 Schedule</b>	<b>22</b>
<b>10 Picture</b>	<b>23</b>

# 1 Overview

The Catapult game is a game with two to eight players in shape of either tanks or catapults. The catapults/tanks are placed in a world, which is as large as the computer screen. The goal of the game is to destroy the other players catapults/tanks by firing ammunition towards them. The catapults/tanks can move all over the world. When a player wants to fire his catapult/tank it has to face the direction towards one of the opponents and the player has to adjust the power of the shot by holding down a key. The shoot is then fired when the key is released. If the player is lucky the shoot will hit the opponents catapult/tank and give it some damage. The damage can be more or less depending on how precise the shoot was. The further away the player is from the opponent the more difficult it is to make a perfect shoot. When a certain limit of damage is reached the catapult/tank is not functional anymore and the game is over. When a catapult/tank is damaged it is less precise and therefore more it is more difficult to make a perfect shoot against the opponents catapult/tank. After a while the catapult will repair it self and be precise again, if it is not being hit by a new shoot from an opponents catapult/tank.

To get ammunition to fire with the player has to move the catapult/tank around to get ammunition. The player can also obtain new health by collecting special health boxes.

In the world there is also obstructions the catapults/tanks can hide behind so the can avoid to be shoot. It is possible before the game to choose between different worlds and different catapults and tanks.

The game can be played both against computer opponents and human opponents. To control the game the player can choose to use keys or a joystick. The only keys needed is keys to move forward, backward, left and right and a key to fire the shoot.

Depending on the time given to develop the game the game can either be made in 2D or 3D. In 2D the world will be viewed from above whereas in 3D it will be viewed birds perspective from the side.

## 2 The Rules

### 2.1 Basics

The game consists of 2-8 tanks on a playing field. The goal of the tanks is to destroy opposing tanks. The tanks can move, and lob a limited number of munitions at each other. There are different types of tanks which determine the specific behaviours of the tank. Tanks can either be controlled by a player or an AI. There are also various field types which effect the tanks differently. Damage done by the explosion is based on how close the explosion was to tank. Tanks will also be knocked away from the explosion, potentially push a tank onto a trap. As tanks get more damaged, it will be hard to control as it will be harder to move, and harder to fire accurately and rapidly. Tanks can reperate health when they are very damaged. To speed up to process, they can restrain from firing. Powerups are also regenerated at random, and can range from health to ammo to limited invulnerability.

## 2.2 Various modes of play

Before the game starts, the player can set the type of game they are going to play. The options they can set are as follows.

**Team or Solo** Selects if game is a free for all, or team play. In teamplay, members of the same team cannot hurt each other. If team play is selected, the players must also select a team to play on.

**Timed or kill limit** Selects the factor to end a round. A round can either end after a certain time, with the player/team with the most kills declared as the winner, or ends once a player/team has a certain number of kills.

**Number of rounds** Selects the number of rounds a team or player must win for the game to end. The first player to win these number of rounds wins.

**limited fire** Reduces the fire rate by a large amount. Makes the game rely more on pushing people of the edge of the field.

## 2.3 The tanks

The player can move the tanks so that the rate of rotation is independent of the speed of the tank. The player can control the horizontal direction (in world terms, not view terms) of the turret, but not the vertical direction of the turret. The direction the turret is facing is independent from the direction the body of the tank, ie the tank will be always aiming in the same direction no matter how the tank rotates.

To fire, the player holds down the fire button. While the fire button is pressed a power meter will grow. The power meter will determine how far the will fire. Prior to hitting the fire button, a player can hold down the charge button to make a shot more powerful. Once the charge button is released, the charge will slowly dissipate unless the player hits the fire button, at which point the charge will stay constant. It is impossible to both increase charge and fire at the same time.

There are various types of tanks that can be chosen from. Various tanks can differ in various

- Speed of movement and rotation and rotating the turret
- Type of movement
  - Tank
  - Freeform
- Type of aiming
  - Turret
  - Fixed
- Amount of damage a shot does
- Rate of fire
- Maximum range of fire

- Damage capacity
- Penalties from damage
  - accuracy
  - rate of movement
  - Rate of fire
- Regeneration rate
- Explosive force from weapons fire
- Mass which controls resistance from explosive force, resistance to pushing, and slippery surfaces

## 2.4 The Playing fields

The playing fields are flat when it comes to the movement of the tanks. However, there may be obstructions with height that can block shots. Obstructions are placed on the map in advance and are always static. There is no dynamic creation of playing fields, only dynamic placement of powerups. Each playing field specifies these qualities.

- Shape and size of the field
- Tileset used for field
- Number of permissible players
- Location of obstructions
- Powerups, and probably distribution for spawning locations across the map
- Type of terrain at various locations

There are various tilesets, some of which have special gameplay properties. For example, some tilesets may have a number of tiles which are slippery. Tilesets also specify the obstructions used. Below are the various tilesets, and their descriptions

**Standard Concrete** Nothing special

**River** Grass with water tiles. The water tiles slow down movement

**Ice** Ice blocks are very slippery. It makes it hard for tanks to accelerate or decelerate under their own power. Makes Pushback very powerful

**Glass** Glassblocks get damaged with direct hits. Enough direct hits and the glass break, leaving a deadly pit. After a few seconds, the hole will repair itself.

## 2.5 Powerups

Powerups randomly appear on the playing field. Pickups trigger an event once they are picked up. Powerups can be damaged, and will be destroyed by 1 direct hit. Upon destruction, they can trigger an event as well. They have a limited lifetime, and will begin to blink when nearing the end of their lifetimes. Powerups will disappear upon retrieval, destruction, and after a certain time. Aside from health and ammo, which are always required to spawn, a playing field can spawn any powerups from this list.

**Speedup** Increase speed

**Fever** Increased rate of fire, as well as multiple shots fired at a time

**Mine** Explodes, causing damage on retrieval. Also explodes on destruction

**Small mode** Turns tank small, making it harder to hit and faster, but does less damage, and cannot shoot as far

**Pusher mode** Tank gets bigger, faster, and can push other robots around easily. The only problem is that extra mass makes it much harder for the tank to stop

**Big Fire** Shots cause more damage, and have a bigger explosion radius

## 3 AI for the game

As a part of the game we are planning to implement an AI. The actions for the AI will be the following:

Shoot

Move (up, down, left and right)

Turn (left, right)

The AI will have the following knowledge about the world:

Opponents position

Position of obstructions

Position of Armor

To get an idea of how to build a good AI for this world, we will start by implementing a simple AI using production rules. The program will start with the first production rule and if it is to be satisfied the corresponding action will be taken. Here is the production rules for a very simple AI and a bit more advanced AI.

### 3.1 Very simple AI

Armor  $\wedge$  Aim  $\rightarrow$  ShootNearestOpponent

Armor  $\rightarrow$  Turn

Move to get armor

This has to be read in the following way. If the AI is in possession of armor and got the aim straight against the opponent, the AI will shoot. If the AI is only in possession of armor, the AI will turn so the aim is against the opponent. Finally if the AI is without armor the AI will move to get armor.

### 3.2 More advanced AI

Armor  $\wedge$  Aim  $\wedge$  CloseToOpponent  $\rightarrow$  ShootOpponent

Armor  $\wedge$  Aim  $\rightarrow$  MoveTowardsOpponent

Armor  $\rightarrow$  Turn

Move to get armor

This AI is different in the way that it is also taking into account that the opponent could be too faraway so it's not possible to make a good shoot. The AI will then move closer to the opponent.

To make the AI more human we will add some probability into the AI. When the AI is making a shoot-action there will be a certain probability that the AI will hit or miss the shot. This can be used to make a easy, medium and hard AI in a simple way.

When the AI has to get armor. We will make sure that the AI knows where the closest armor is located. First we will just let the AI go to the armor which is closest when you draw a straight line from the AI to the armor, but we will also implement some more advanced path finding if it should be the case that it is very important to be able to get armor as quick as possible. This method will take the obstructions into account.

Another improvement we will do after implementing the simple AI's is to make the AI hide behind an obstruction if it is damaged from a shoot and needs to gain new power.

## 4 Artwork and user interface

### 4.1 Artwork

Artwork will be created using Photoshop and 3d Studio Max. 3D models will be created to create either 3d or 2d images for the game. Photoshop will be used to create skins and textures for the environments.

Depending on the theme of the game, either a realistic or comic feel will be used in creating the artwork for the game. For the realistic approach, textures will be created from photos of landscapes and tank miniatures to create a real-world feel to the game or simply a miniature tank set. For the comic approach, hand-drawn images will be scanned in to create textures or programs such as Painter 7 or Photoshop will be used to create designs and textures over the 3d models.

### 4.2 User Interface

The user interface of the game will consist of indicators for each player:

## Vehicle Status

**Damage rating** The amount of sustained damage for each player will be listed transparently on the bottom of the screen. Damage is displayed in percentage form, with 0% being the least damaged, and 100% being the most damaged

**Power Gauge for weapon** Is displayed as a semicircle around the Tank. As the power grows, the arc grows, creating a full circle at full power.

**Time Remaining** The time remaining is displayed in the upper right corner

Also a menu will be available when a character pauses the game. The menu will consist of :

**Options** allows players to change their controls and adjust in game features.

**Quit** allows players to leave the game.

## 5 Sketches (to be finished)

## 6 Music and Sound

### 6.1 Music

Music will be created from sampled loops of warfare noises such as explosions and gunfire to create a soundtrack that is high energy and captures the adrenaline of being in a warfare situation. Soundfoundry Acid Pro and Soundry will be used to create beats and edit tracks.

### 6.2 Sound Effects

Sound effects will be created from recorded sounds, captured on a microphone. This will give the game a more realistic feel than if the games sound were computer generated or merely taken from other games. The same programs mentioned in the music section will be used to create the sound effects.

## 7 Unit Descriptions

Catapult: (All around vehicle) - 32 pts.

Range - 7

Damage - 6

Accuracy - 7

Speed - 6

Armor - 6

Attacks in facing direction.

Vehicular movement.

Damage consequences: Moves slower, and loses range. (Neither by much)

M1 Abrams: (Heavily armored and lumbering) - 32 pts.

Range - 6

Damage - 7

Accuracy - 6  
Speed - 4  
Armor - 9  
Attacks in aimed direction.  
Vehicular movement.  
Damage consequences: Moves slower, loses aiming precision, loses range.

Mortar team: (Fast, accurate, and weak) - 31 pts.  
Range - 6  
Damage - 6  
Accuracy - 8  
Speed - 8  
Armor - 3  
Attacks in aimed direction, and has to stop to fire.  
8-way movement.  
Damage consequences: Moves slower.

Howitzer: (Most heavily armored, long ranged, and most lumbering) - 34 pts.  
Range - 10  
Damage - 6  
Accuracy - 5  
Speed - 3  
Armor - 10  
Attacks in facing direction.  
Vehicular movement.  
Damage consequences: Loses range.

Grenadier: (Fastest, most precise, and weakest) - 31 pts.  
Range - 4  
Damage - 6  
Accuracy - 9  
Speed - 10  
Armor - 2  
Attacks in aimed direction.  
8-way movement.  
Damage consequences: Moves slower, loses range and accuracy.

## 8 Technical Specs

### 8.1 Platform

The game will run exclusively on the Windows platform. We will use Visual C++ 6.0 and DirectX SDK 8.1 as our compiler and library of choice. The target hardware will be an Intel Pentium 100 Mhz or better running at least Windows 98.

### 8.2 Game Engine

After initialization, the game engine enters a loop. The steps in the loop are as follows.

1. Check control signals (Such as end game, pause)
2. Poll input modules for input
3. Compute new game state
4. Send gamestate to listeners

Each loop is called a tick. The game is set to run at 30 ticks a second.

### 8.2.1 Tank Behaviour

Each tank is internally represented using a Tank data structure.

```
struct Tank {
short id;           //Unique ID
int x, y;          //Position
int dx, dy;        //Velocity
float direction;   //Facing direction
int life;          //Remaining life
int charge;        //The number of turns charged
int powerups[];    //Lifetime of powerups. Powerups with zero life means
                  //powerup is not active
};
```

### 8.2.2 Collision detection

Collision detection will be bounding sphere and box based. Robots will be spheres with a diameter of it's diagonal. Object which collide with transfer each others force to each other, with the exception of static objects, which always return an equal force as being applied to it.

### 8.2.3 Output Interface

Each output interface must implement a function which takes in a GameData object.

```
struct GameData {
short tank_count;   //Number of tanks
short bullet_count; //Number of bullets
short tile_count;   //Number of terrain changes
Tank tanks[];      //Array of tank data
Bullet bullets[];   //Array of bullet data
Tile tiles[];       //Array of tile changes
};
```

Examples of output modules are graphical displays, network clients, and AI players.

### 8.2.4 Input Interface

Each input module must implement only one function, move, which takes a player.id and returns a TankMove data structure. Move is called every game tick.

```

struct TankMove {
short up;          //Control up or forward movement rate depending on
                  //movement type. Negative values specifies opposite
                  //direction.
float right; //Control right movement or right turning rate depending
            //on movement type. Negative values specifies opposite
            //direction.
float turret;    //Controls rotation of turret. Positives values
                //specifies right rotation, negative left.
bool charge;    //Specifies if the tank should be charging a shot. To
                //shoot, the module must set this value to false a tick
                //after it has been true.
};

```

Examples of Input modules are local players(keyboard, gamepads), network players , or AI players.

### 8.3 Tank Specification

Each type of tank has it's own file specifying tank behaviour. All tank files will be stored the same directory. Each property is a separate line in order in the config file. Lines that start with a // will be ignored.

**id** A unique number identifying the tank

**name** Name of the tank

**movement type** Determines movement type of the tank.

**width** Describes the width of the tank's bounding box

**height** Describes the height of the tank's bounding box

**turret location** Describes the location of the turret pivot in relation to the back left of the tank body.

**body graphic** The file where the body graphics are stored

**turret graphic** The file where the turret graphics are stored

**turret pivot** The location of the pivot for the turret from the bottom left corner or the graphic.

**shot graphic** The file where the shot graphics are stored

**damage multiplier** The amount of damage a tank will do compared to a base tank.

**health multiplier** How much life a tank has compared to a base tank

**charge multiplier** How far a tank will shoot compared to a base tank

**cooldown time** How many ticks after shooting, can a tank begin charging again.

**speed multiplier** How fast the tank can move compared to a base tank.

**turn multiplier** How fast the tank turns compared to a base tank. This is ignored for tanks that do not turn, but still must be specified.

**turret speed multiplier** The maximal turn rate of the turret compared to a base tank.

**mass multiplier** Mass determines how resistant a tank is to change in velocity. This is significant in slippery surfaces, standard acceleration, getting hit, and pushing matches.

**damage speed** Speed of the tank at full damage

**damage aim offset** Amount to offset shots at full damage

**damage cooldown** Cooldown period at full damage

## 8.4 Tileset Specification

Each tileset will be described in a file, and all tileset files will be stored in the same directory. The first line in the file will specify the image file where each tile is stored. Tiles are indexed from left to right, then from top to bottom, with the upper left tile being 0. The next line contains the unique tilemap id.

Then the settings for each tile are set. The first number specifies the tile number. The next line has the block elevation. Every line after that contains options for the block, one option per block. The options are

slow Reduces the speed of tanks by 1/2

slippery Reduces traction of tanks

pit Instant death for any tank

obstacle Cannot be moved onto

Each tile specification occurs after a blank line, and has no blank lines in it. Tiles without any specification are set to elevation 0 with no options.

Lines that start with `//` are ignored.

## 8.5 Level Specification

Each level will be specified by an individual file. Maps are made of square tiles. Every level must specify these components.

**size** The number of vertical and horizontal tiles used in the map

**maxplayers** The maximum number of players allowed on the map

**tileset** Specifies a set of tiles from which to build the level with

**map layout** Specifies the tile layout to form the level.

**obstruction layout** Specifies which tiles are obstructions the map

**starting points** Specifies where each tanks starts at

**powerup layout** Specifies the probability of a powerup appearing in a tile.

In the level file, lines that start with a // will be ignored. The first non blank or commented line should contain four space delimited numbers. The first and second should specify the number of horizontal and vertical tiles on the map respectively. The third number should specify the maximum number of players allowing on the map. The fourth number should specify the tileset.

The next lines should describe which tiles will be used to make up the level layout, each line describing a horizontal row in the level. Each tile will specify its contents by referring to numbered tiles in the tileset.

Then after that, the next lines lay out the obstruction layers. These are things like trees or glass blocks which tanks cannot move through. For example a very simple and small level would look like this.

Next, we specify where the start locations are. Each location has a separate line. There are as many starting locations as there are max players. The first line represents the starting point of the first player, and the second line represents the starting point of the second player, and so on. Each line only consists of two numbers, start col, start row. Every location is specified 0 indexed with 0 0 being the upper left corner.

Lastly we specify the squares where powerups can occur. Each line describes a specific powerup. Powerups not listed are assumed to generate themselves over evenly over all squares which can be moved on. Each line starts out with a power up id and then repeating tile locations and probability density. The probability of a powerup spawning in a particular tile is density/overall density.

```
// width height players tileset
3 4 4 1

//Background
1 1 1
1 2 1
1 2 1
1 1 1

//Obstructions
0 0 0
1 0 1
1 0 1
0 0 0

//Starting points
0 0
2 0
0 3
2 3

//MegaPowerup in center
3 1 1 1 1 2 1
```

## 8.6 Graphics Engine

We will be using a 2D/3D graphics engine that was included with the book *Programming Role-Playing Games w/ DirectX 8.0* by Jim Adams. The engine basically sits directly above DirectX and abstracts all of the tedious details away.

It was designed so that the user won't need to learn DirectX before they can use the engine. It provides all of the necessary modules for game development like system, graphics, input, sound, and network.

### 8.6.1 System Module

The System Module contains the class definition of `cApplication`. The main purpose of the module is to abstract away the windowing system. The user doesn't need to deal with window creation and message loops if they don't want to. All they need to do is create a new class derived from `cApplication` and override a few member methods.

```
class cApplication
{
private:
    HINSTANCE    m_hInst;
    HWND        m_hWnd;

protected:
    char        m_Class[MAX_PATH];
    char        m_Caption[MAX_PATH];

    WNDCLASSEX  m_wcex;

    DWORD       m_Style;
    DWORD       m_XPos;
    DWORD       m_YPos;
    DWORD       m_Width;
    DWORD       m_Height;

public:
    cApplication();

    HWND        GethWnd();
    HINSTANCE    GethInst();

    BOOL Run();
    BOOL Error(BOOL Fatal, char *Text, ...);

    BOOL Move(long XPos, long YPos);
    BOOL Resize(long Width, long Height);

    BOOL ShowMouse(BOOL Show = TRUE);

    virtual FAR PASCAL MsgProc(HWND hWnd, UINT uMsg,
                               WPARAM wParam, LPARAM lParam);
    virtual BOOL Init()        { return TRUE; }
    virtual BOOL Shutdown()    { return TRUE; }
    virtual BOOL Frame()       { return TRUE; }
};
```

## 8.6.2 Graphics Module

This is the main module of the BookCode Engine. It abstracts the tedious details of initializing and setting up DirectX. At the heart of the Graphics Modules is the `cGraphics` class definition. Some of its primary methods are `Init`, `Shutdown`, `SetMode`, `BeginScene`, `EndScene`, and `Display`.

```
class cGraphics
{
protected:
    HWND          m_hWnd;
    IDirect3D8    *m_pD3D;
    IDirect3DDevice8 *m_pD3DDevice;
    ID3DXSprite   *m_pSprite;

    D3DDISPLAYMODE m_d3ddm;

    BOOL          m_Windowed;
    BOOL          m_ZBuffer;
    BOOL          m_HAL;

    long          m_Width;
    long          m_Height;
    char          m_BPP;

    char          m_AmbientRed;
    char          m_AmbientGreen;
    char          m_AmbientBlue;

public:
    cGraphics();
    ~cGraphics();

    IDirect3D8    *GetDirect3DCOM();
    IDirect3DDevice8 *GetDeviceCOM();
    ID3DXSprite   *GetSpriteCOM();

    BOOL Init();
    BOOL Shutdown();

    BOOL SetMode(HWND hWnd, BOOL Windowed = TRUE, BOOL UseZBuffer = FALSE,
                long Width = 0, long Height = 0, char BPP = 0);
    long GetNumDisplayModes();
    BOOL GetDisplayModeInfo(long Num, D3DDISPLAYMODE *Mode);
    char GetFormatBPP(D3DFORMAT Format);
    BOOL CheckFormat(D3DFORMAT Format, BOOL Windowed, BOOL HAL);

    BOOL Display();

    BOOL BeginScene();
    BOOL EndScene();
};
```

```

    BOOL BeginSprite();
    BOOL EndSprite();

    BOOL Clear(long Color = 0, float ZBuffer = 1.0f);
    BOOL ClearDisplay(long Color = 0);
    BOOL ClearZBuffer(float ZBuffer = 1.0f);

    long GetWidth();
    long GetHeight();
    char GetBPP();
    BOOL GetHAL();
    BOOL GetZBuffer();

    BOOL SetPerspective(float FOV=D3DX_PI / 4.0f, float Aspect=1.3333f,
                       float Near=1.0f, float Far=10000.0f);

    BOOL SetWorldPosition(cWorldPosition *WorldPos);
    BOOL SetCamera(cCamera *Camera);
    BOOL SetLight(long Num, cLight *Light);
    BOOL SetMaterial(cMaterial *Material);
    BOOL SetTexture(short Num, cTexture *Texture);

    BOOL SetAmbientLight(char Red, char Green, char Blue);
    BOOL GetAmbientLight(char *Red, char *Green, char *Blue);

    BOOL EnableLight(long Num, BOOL Enable = TRUE);
    BOOL EnableLighting(BOOL Enable = TRUE);
    BOOL EnableZBuffer(BOOL Enable = TRUE);
    BOOL EnableAlphaBlending(BOOL Enable = TRUE, DWORD Src = D3DBLEND_SRCALPHA,
                             DWORD Dest = D3DBLEND_INVSRCALPHA);
    BOOL EnableAlphaTesting(BOOL Enable = TRUE);
};

```

### 8.6.3 Input Module

The Input Module can handle 3 different input devices. It supports the keyboard, mouse, and joystick devices. The module defines two primary classes, `cInput` and `cInputDevice`. The `cInput` class is basically a wrapper that initializes the `IDirectInput8` interface. You must initialize this interface before DirectX allows you to create any `IDirectInputDevice8` interfaces. The `cInputDevice` is the actual input device. All input reading will be handled by this class.

```

class cInput
{
protected:
    HWND          m_hWnd;
    IDirectInput8 *m_pDI;

public:
    cInput();
    ~cInput();
};

```

```

    IDirectInput8 *GetDirectInputCOM();
    HWND          GethWnd();

    BOOL Init(HWND hWnd, HINSTANCE hInst);
    BOOL Shutdown();
};

class cInputDevice
{
public:
    cInput          *m_Input;

    short           m_Type;
    IDirectInputDevice8 *m_pDIDevice;

    BOOL           m_Windowed;

    char           m_State[256];
    DIMOUSESTATE  *m_MouseState;
    DIJOYSTATE    *m_JoystickState;

    BOOL           m_Locks[256];

    long           m_XPos, m_YPos;

    static BOOL FAR PASCAL EnumJoysticks(LPCDIDEVICEINSTANCE pdInst, LPVOID pvRef);

public:
    cInputDevice();
    ~cInputDevice();

    IDirectInputDevice8 *DeviceCOM();

    // Generic functions - all devices
    BOOL Create(cInput *Input, short Type, BOOL Windowed = TRUE);
    BOOL Free();

    BOOL Clear();
    BOOL Read();
    BOOL Acquire(BOOL Active = TRUE);

    BOOL GetLock(char Num);
    BOOL SetLock(char Num, BOOL State = TRUE);

    long GetXPos();
    BOOL SetXPos(long XPos);
    long GetYPos();
    BOOL SetYPos(long YPos);
    long GetXDelta();
    long GetYDelta();

```

```

// Keyboard specific functions
BOOL GetKeyState(char Num);
BOOL SetKeyState(char Num, BOOL State);
BOOL GetPureKeyState(char Num);
short GetKeyPress(long TimeOut = 0);
long GetNumKeyPresses();
long GetNumPureKeyPresses();

// Mouse/Joystick specific functions
BOOL GetButtonState(char Num);
BOOL SetButtonState(char Num, BOOL State);
BOOL GetPureButtonState(char Num);
long GetNumButtonPresses();
long GetNumPureButtonPresses();
};

```

#### 8.6.4 Sound Module

In the sound module, there are three main wrapper classes, `cSound`, `cSoundChannel`, and `cMusicChannel`. These three classes work together to allow the user to load and play any `.wav` and `.mid` files. The `cSoundChannel` class is designed to handle `.wav` files. These files are used for sound effects. The `cMusicChannel`, on the other hand, is designed to handle `.mid` files. These files are intended to be background musics.

```

class cSound
{
protected:
// Sound system related
HWND    m_hWnd;

// Master volume level
long    m_Volume;

// Playback format settings
long    m_Frequency;

// Music and sound objects
IDirectMusicPerformance8 *m_pDMPPerformance;
IDirectMusicLoader8      *m_pDMLoader;

public:
cSound();
~cSound();

// Functions to retrieve COM interfaces
IDirectMusicPerformance8 *GetPerformanceCOM();
IDirectMusicLoader8      *GetLoaderCOM();

// Init and shutdown functions
BOOL Init(HWND hWnd, long Frequency = 22050);
BOOL Shutdown();

```

```

        // Volume get/set
        long GetVolume();
        BOOL SetVolume(long Percent);
};

class cSoundChannel
{
    friend class cSound;

protected:
    cSound          *m_Sound;

    IDirectMusicAudioPath *m_pDMPATH;
    IDirectMusicSegment8 *m_pDMSegment;

    long            m_Volume;
    long            m_Loop;

public:
    cSoundChannel();
    ~cSoundChannel();

    IDirectMusicAudioPath8 *GetAudioPathCOM();
    IDirectMusicSegment8 *GetSegmentCOM();

    BOOL Create(cSound *Sound);
    BOOL Free();

    BOOL Play(char *Filename, long VolumePercent = 100, long Loop = 1);
    BOOL Play(void *Ptr, unsigned long Size, long VolumePercent = 100, long Loop = 1);
    BOOL Stop();

    long GetVolume();
    BOOL SetVolume(long Percent);

    BOOL IsPlaying();
};

class cMusicChannel
{
    friend class cSound;

protected:
    cSound          *m_Sound;
    IDirectMusicSegment8 *m_pDMSegment;
    long            m_Volume;

public:
    cMusicChannel();
    ~cMusicChannel();
};

```

```

    IDirectMusicSegment8 *GetSegmentCOM();

    BOOL Create(cSound *Sound);

    BOOL Load(char *Filename);
    BOOL Free();

    BOOL SetDLS(cDLS *DLS);

    BOOL Play(long VolumePercent = 100, long Loop = 1);
    BOOL Stop();

    long GetVolume();
    BOOL SetVolume(long Percent = 100);

    BOOL SetTempo(long Percent = 100);

    BOOL IsPlaying();
};

```

### 8.6.5 Network Module

The network module is composed of three classes, `cNetworkAdapter`, `cNetworkServer`, and `cNetworkClient`. The `cNetworkAdapter` represents the actual network interface. This could be the loopback device, or a valid network interface. The `cNetworkServer` represents the game server. It handles all incoming connection requests. The `cNetworkClient` represents the game client. It must first connect to the server before establishing a game session.

```

class cNetworkAdapter
{
protected:
    DPN_SERVICE_PROVIDER_INFO *m_AdapterList;
    unsigned long m_NumAdapters;

    static HRESULT WINAPI NetMsgHandler(PVOID pvUserContext, DWORD dwMessageId,
                                        PVOID pMsgBuffer);

public:
    cNetworkAdapter();
    ~cNetworkAdapter();

    BOOL Init();
    BOOL Shutdown();
    long GetNumAdapters();
    BOOL GetName(unsigned long Num, char *Buf);
    GUID *GetGUID(unsigned long Num);
};

class cNetworkServer
{

```

```

protected:
    IDirectPlay8Server *m_pDPSServer;

    BOOL                m_Connected;

    long                m_Port;

    char                m_SessionName[MAX_PATH];
    char                m_SessionPassword[MAX_PATH];

    long                m_MaxPlayers;
    long                m_NumPlayers;

    static HRESULT WINAPI NetworkMessageHandler(PVOID pvUserContext,
                                                DWORD dwMessageId,
                                                PVOID pMsgBuffer);

    virtual BOOL AddPlayerToGroup(DPNMSG_ADD_PLAYER_TO_GROUP *Msg);
    virtual BOOL AsyncOpComplete(DPNMSG_ASYNC_OP_COMPLETE *Msg);
    virtual BOOL ClientInfo(DPNMSG_CLIENT_INFO *Msg);
    virtual BOOL ConnectComplete(DPNMSG_CONNECT_COMPLETE *Msg);
    virtual BOOL CreateGroup(DPNMSG_CREATE_GROUP *Msg);
    virtual BOOL CreatePlayer(DPNMSG_CREATE_PLAYER *Msg);
    virtual BOOL DestroyGroup(DPNMSG_DESTROY_GROUP *Msg);
    virtual BOOL DestroyPlayer(DPNMSG_DESTROY_PLAYER *Msg);
    virtual BOOL EnumHostsQuery(DPNMSG_ENUM_HOSTS_QUERY *Msg);
    virtual BOOL EnumHostsResponse(DPNMSG_ENUM_HOSTS_RESPONSE *Msg);
    virtual BOOL GroupInfo(DPNMSG_GROUP_INFO *Msg);
    virtual BOOL HostMigrate(DPNMSG_HOST_MIGRATE *Msg);
    virtual BOOL IndicateConnect(DPNMSG_INDICATE_CONNECT *Msg);
    virtual BOOL IndicatedConnectAborted(DPNMSG_INDICATED_CONNECT_ABORTED *Msg);
    virtual BOOL PeerInfo(DPNMSG_PEER_INFO *Msg);
    virtual BOOL Receive(DPNMSG_RECEIVE *Msg);
    virtual BOOL RemovePlayerFromGroup(DPNMSG_REMOVE_PLAYER_FROM_GROUP *Msg);
    virtual BOOL ReturnBuffer(DPNMSG_RETURN_BUFFER *Msg);
    virtual BOOL SendComplete(DPNMSG_SEND_COMPLETE *Msg);
    virtual BOOL ServerInfo(DPNMSG_SERVER_INFO *Msg);
    virtual BOOL TerminateSession(DPNMSG_TERMINATE_SESSION *Msg);

public:
    cNetworkServer();
    ~cNetworkServer();

    IDirectPlay8Server *GetServerCOM();

    BOOL Init();
    BOOL Shutdown();

    BOOL Host(GUID *guidAdapter, long Port, char *SessionName,
             char *Password = NULL, long MaxPlayers = 0);
    BOOL Disconnect();

```

```

    BOOL IsConnected();

    BOOL Send(DPNID dpnidPlayer, void *Data, unsigned long Size,
             unsigned long Flags = 0);
    BOOL SendText(DPNID dpnidPlayer, char *Text, unsigned long Flags = 0);

    BOOL DisconnectPlayer(long PlayerId);

    BOOL GetIP(char *IPAddress, unsigned long PlayerId = 0);
    BOOL GetName(char *Name, unsigned long PlayerId);
    long GetPort();
    BOOL GetSessionName(char *Buf);
    BOOL GetSessionPassword(char *Buf);
    long GetMaxPlayers();
    long GetNumPlayers();
};

class cNetworkClient
{
protected:
    IDirectPlay8Client *m_pDPClient;

    BOOL                m_Connected;

    char                m_IPAddress[MAX_PATH];
    long               m_Port;

    char                m_Name[MAX_PATH];

    char                m_SessionName[MAX_PATH];
    char                m_SessionPassword[MAX_PATH];

    static HRESULT WINAPI NetworkMessageHandler(PVOID pvUserContext,
                                               DWORD dwMessageId,
                                               PVOID pMsgBuffer);

    virtual BOOL AddPlayerToGroup(DPNMSG_ADD_PLAYER_TO_GROUP *Msg);
    virtual BOOL AsyncOpComplete(DPNMSG_ASYNC_OP_COMPLETE *Msg);
    virtual BOOL ClientInfo(DPNMSG_CLIENT_INFO *Msg);
    virtual BOOL ConnectComplete(DPNMSG_CONNECT_COMPLETE *Msg);
    virtual BOOL CreateGroup(DPNMSG_CREATE_GROUP *Msg);
    virtual BOOL CreatePlayer(DPNMSG_CREATE_PLAYER *Msg);
    virtual BOOL DestroyGroup(DPNMSG_DESTROY_GROUP *Msg);
    virtual BOOL DestroyPlayer(DPNMSG_DESTROY_PLAYER *Msg);
    virtual BOOL EnumHostsQuery(DPNMSG_ENUM_HOSTS_QUERY *Msg);
    virtual BOOL EnumHostsResponse(DPNMSG_ENUM_HOSTS_RESPONSE *Msg);
    virtual BOOL GroupInfo(DPNMSG_GROUP_INFO *Msg);
    virtual BOOL HostMigrate(DPNMSG_HOST_MIGRATE *Msg);
    virtual BOOL IndicateConnect(DPNMSG_INDICATE_CONNECT *Msg);
    virtual BOOL IndicatedConnectAborted(DPNMSG_INDICATED_CONNECT_ABORTED *Msg);
};

```

```

    virtual BOOL PeerInfo(DPNMSG_PEER_INFO *Msg);
    virtual BOOL Receive(DPNMSG_RECEIVE *Msg);
    virtual BOOL RemovePlayerFromGroup(DPNMSG_REMOVE_PLAYER_FROM_GROUP *Msg);
    virtual BOOL ReturnBuffer(DPNMSG_RETURN_BUFFER *Msg);
    virtual BOOL SendComplete(DPNMSG_SEND_COMPLETE *Msg);
    virtual BOOL ServerInfo(DPNMSG_SERVER_INFO *Msg);
    virtual BOOL TerminateSession(DPNMSG_TERMINATE_SESSION *Msg);

public:
    cNetworkClient();
    ~cNetworkClient();

    IDirectPlay8Client *GetClientCOM();

    BOOL Init();
    BOOL Shutdown();

    BOOL Connect(GUID *guidAdapter, char *IP, long Port, char *PlayerName,
                char *SessionName, char *SessionPassword = NULL);
    BOOL Disconnect();
    BOOL IsConnected();

    BOOL Send(void *Data, unsigned long Size, unsigned long Flags = 0);
    BOOL SendText(char *Text, unsigned long Flags = 0);

    BOOL GetIP(char *IPAddress);
    long GetPort();
    BOOL GetName(char *Name);
    BOOL GetSessionName(char *Buf);
    BOOL GetSessionPassword(char *Buf);
};

```

## 8.7 Version Control

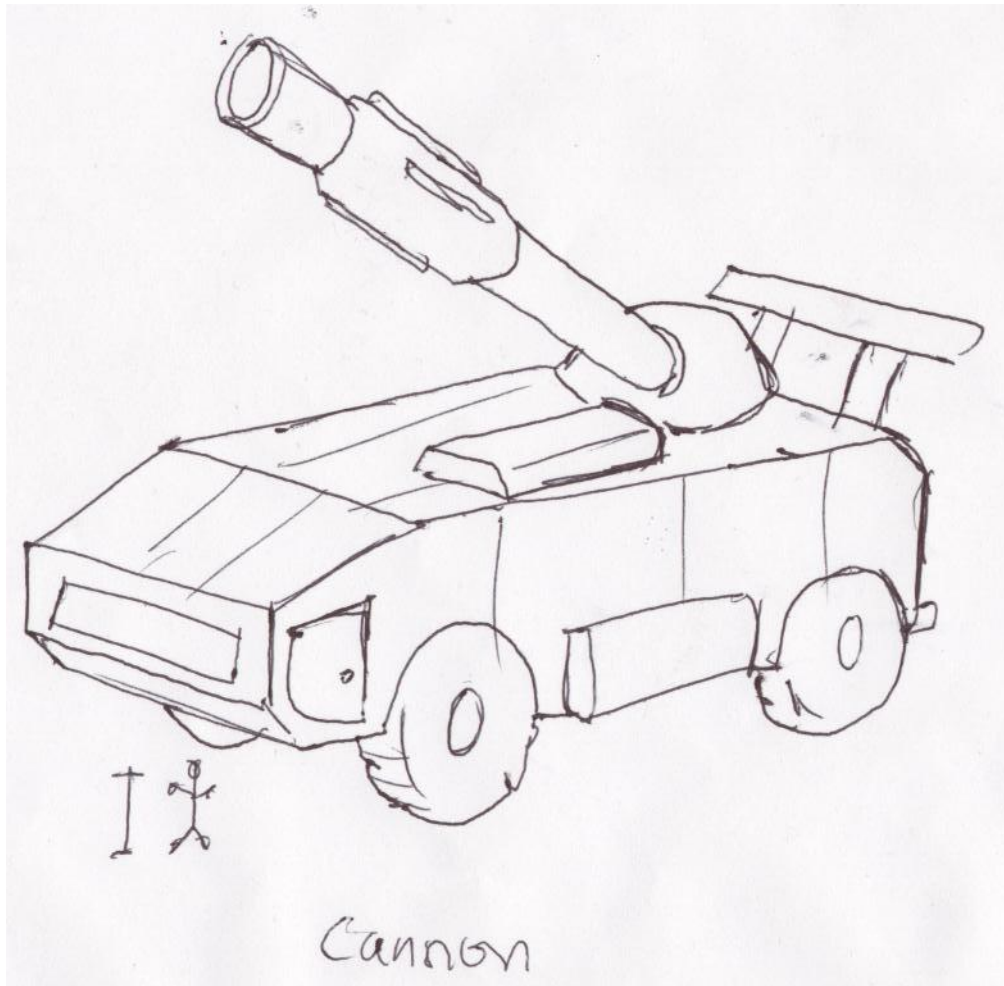
We have a dedicated CVS<sup>1</sup> server where we still store all of our version controls. The hostname of the CVS server is *mandatoryexplosions.com*. The server is setup to accept only incoming `ssh` connections so the `pserver` authentication method is disabled. Everybody who require access to the server will be given an individual user account. Please email Charlie (*chui@mandatoryexplosions.com*) if you need access to the CVS server.

## 9 Schedule

We have the following work areas to take care of in the process of implementing the game.

---

<sup>1</sup>Quick-start guide: <http://www.gentoo.org/doc/en/cvs-tutorial.xml>



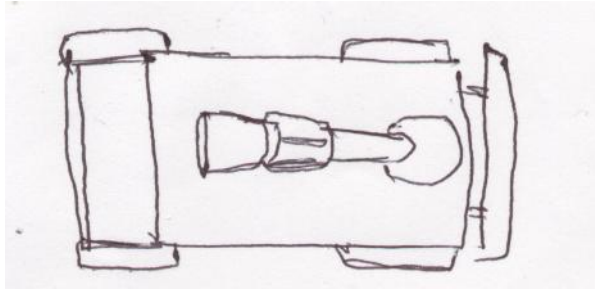
AI	Anders
Game-engine	Charlie
Graphic-engine	Shaun/Hieu
Art/Music	Bob (help from Charlie)
Network	Shaun/Hieu
Control Input	Shaun/Hieu

The current schedule for the game is the following:

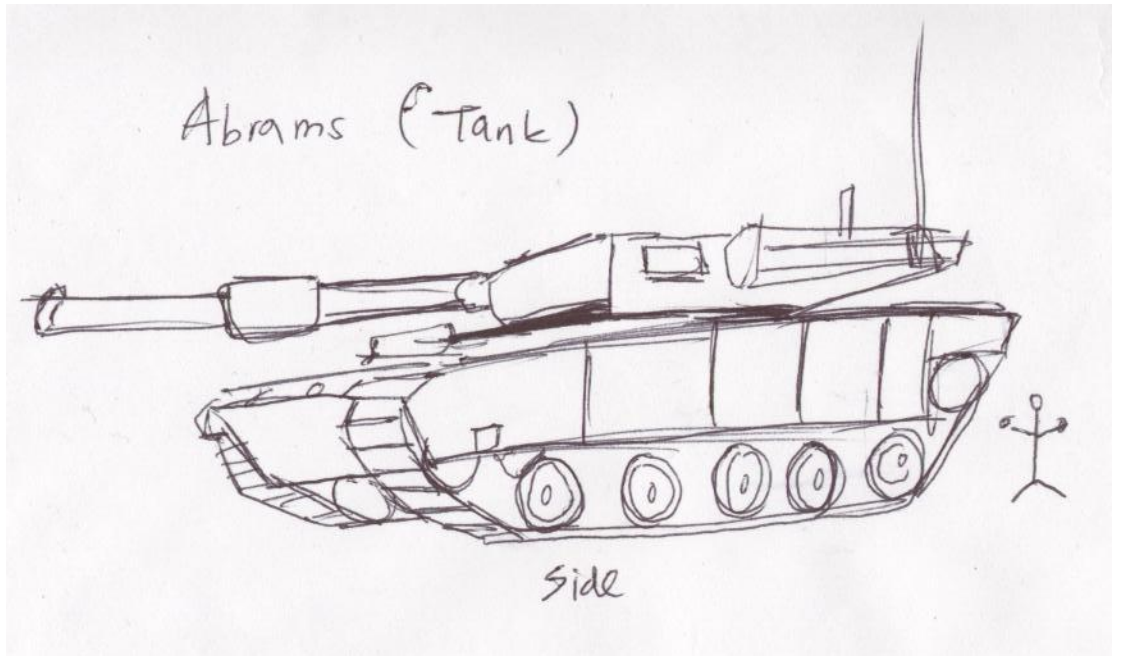
Week 5	Interfaces, Control Input and develop design document
Week 6	Design Document due
Week 7	Prototype I (without Art, Network, AI)
Week 8	Prototype II (with Art, AI)
Week 9	Prototype III (with Network, Music)
Week 10	Testing

## 10 Picture

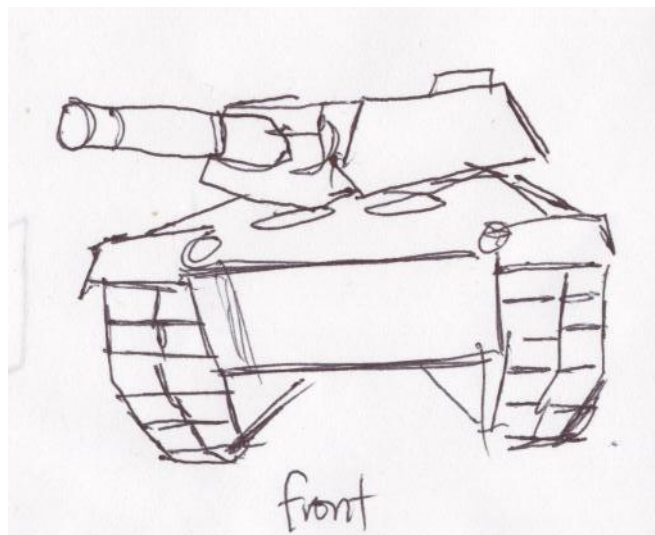
Pictures go here. They should be up soon, the artwork is done



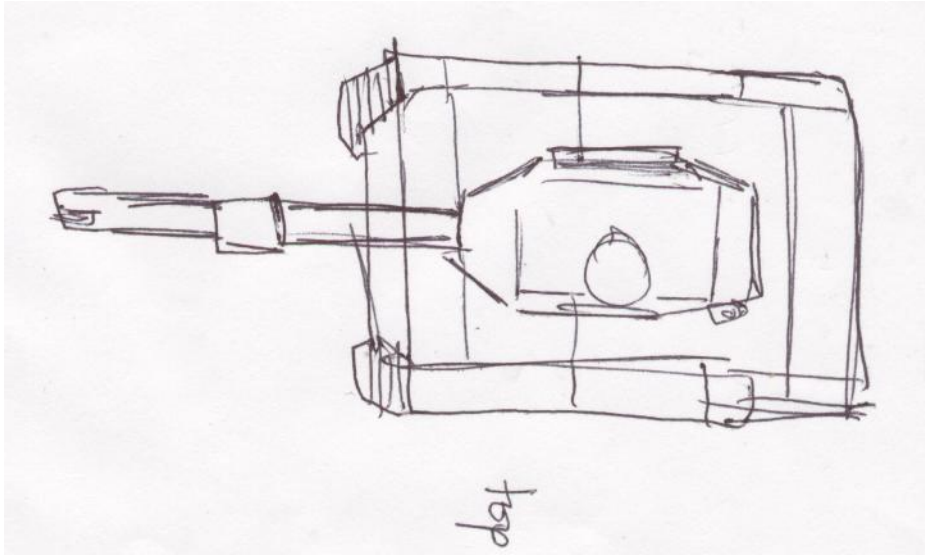
001.jpg



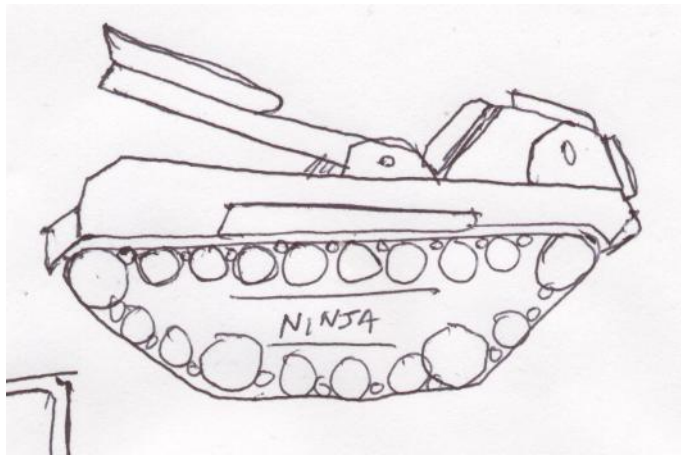
002.jpg



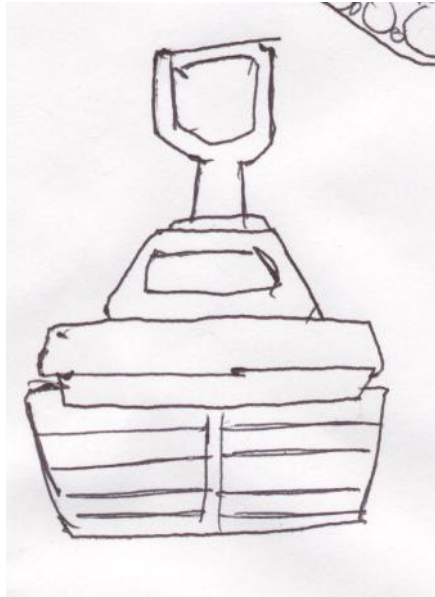
003.jpg



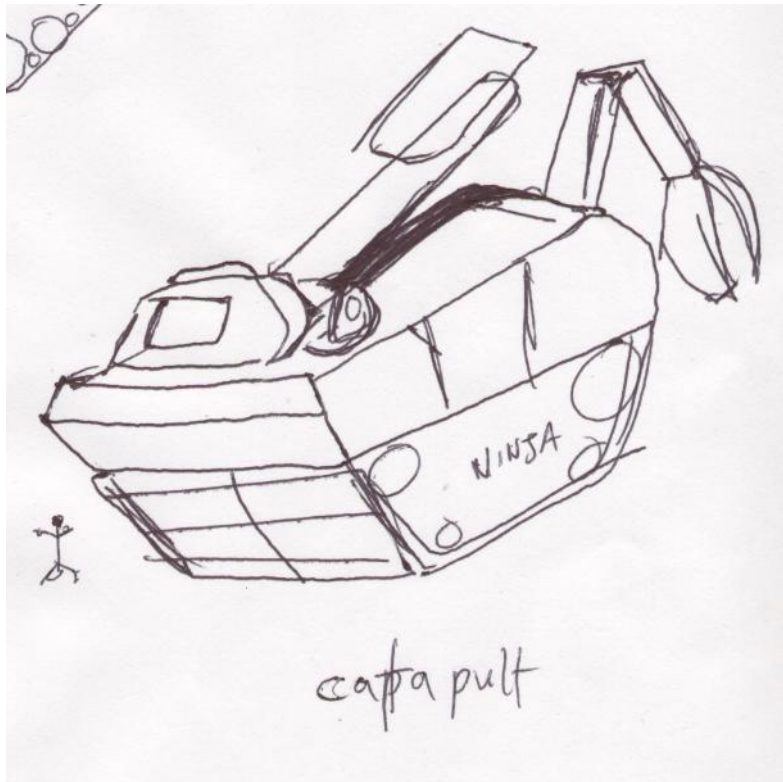
004.jpg



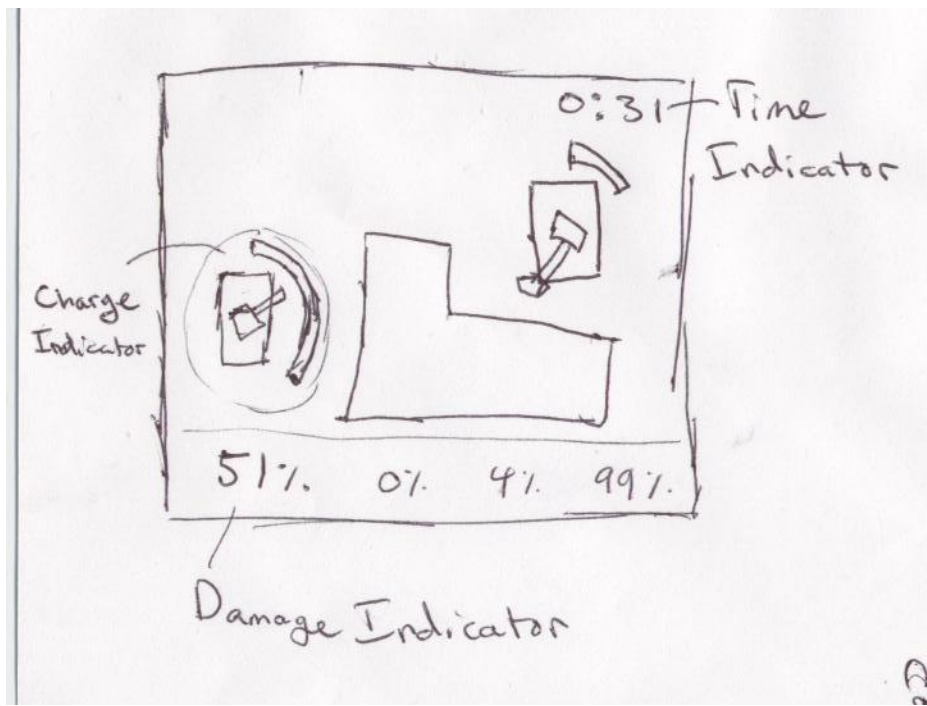
005.jpg



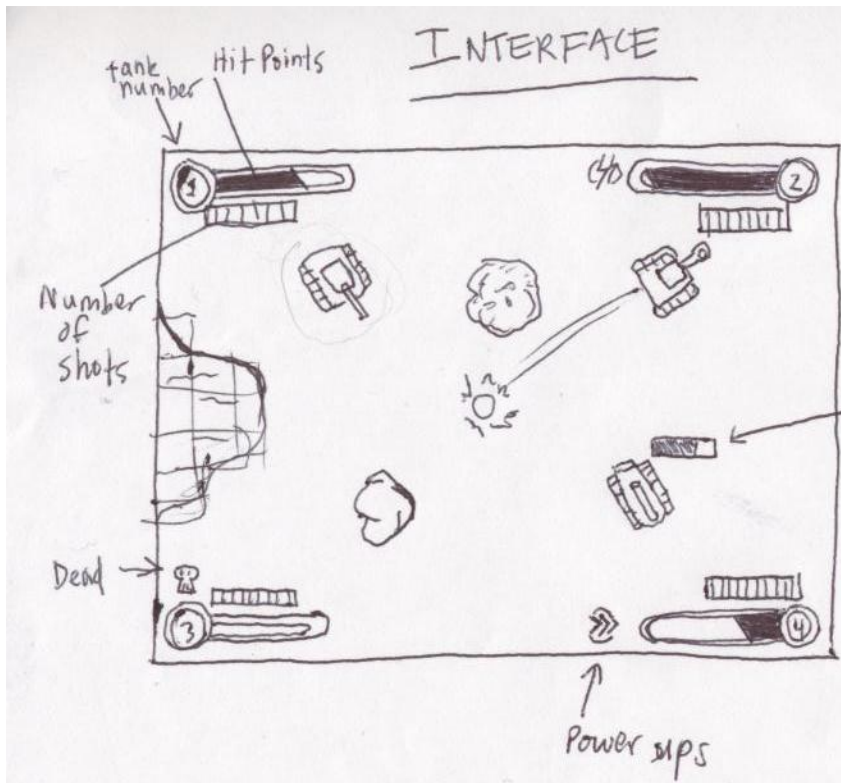
006.jpg



007.jpg



008.jpg



### Obstacles

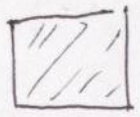
Rocks, concrete structures



River (Water)



Ice



Glass



Feve  
Speed  
Extra  
Speed  
Mine  
Small  
Push  
Big F