Club Sandwich

# TECHNICAL DOCUMENTATION

Desktruction Derby:

Michael Brown – mjbrown@live.com.au
Paul Watkins – watkinzez@hotmail.com
Kitman Yiu – kitman200220022002@gmail.com

10/18/2013

# TABLE OF CONTENTS

# 1 COMPONENT LIST
(DETAILED DESCRIPTION OF GAME COMPONENTS)

User: Programmer

## Menu

### Menu State
Responsible for updating and drawing all the objects in the menu. From here the user controls various graphical and control options, chooses to start a new game, review high scores, or exit the game. The Menu State will be also be inherited from the Base State.

## Game

### Game State
Covers the main game loop. Updates all assets and objects related to the game (player movement, NPC movement, collisions, graphical effects, audio etc), and then draws relevant content to the screen.

### Camera
Follows an entity and sets position of all objects in the level relative to that entity. If not connected to an entity, the camera defaults to (0,0,0).

### AI path finding
Abstract class for all in-game objects to inherit from. Contains a position, a model and a bounding box. Requires implementation of an update and draw function.details we be represent In the LEVEL DETAILS

### Scenery
Inherits from Entity. Inactive objects within the level that act as part of the level (walls, obstacles etc).

### Vehicle
Abstract class that inherits from Entity. Represents all vehicles in the game. Contains velocity and other related momentum data, as well as damage to the vehicle. Also implements collision logic; for each update the given vehicle checks for collisions with other entities within its local area.

### Player
Inherits from Vehicle. Represents the player vehicle. Contains a control object.

### NPC
Abstract class that inherits from Vehicle. Represents the NPC vehicles. Contains an AI state to keep track of the current AI being used, methods for the different base AI states (wander, seek, avoid). Requires implementation of a berserk function. we be represent in the LEVEL DETAILS

### Katherine
Inherits from NPC. Represents the Katherine NPC class. we be represent in the LEVEL DETAILS

### Swarmer
Inherits from NPC. Represents the Swarmer NPC class. We be represent in the LEVEL DETAILS

### Splitter
Inherits from NPC. Represents the Splitter NPC class. We be represent in the LEVEL DETAILS

### Cloaker
Inherits from NPC. Represents the Cloaker NPC class. we be represent in the LEVEL DETAILS

### Enemy
The Enemy class responsible is to control how the AI should be move and what behaviours that the enemy is the enemy also have responsible to detect did it collider with other enemies or player or wall. the pseudo code of enemy class we be represent in the LEVEL DETAILS

### Rule Handler

Stores the logic of each of the rules, as well as which rules are currently active. Rules can be made active or inactive. Player data is sent whenever an NPC is destroyed to check against the currently active rules, and returns a value indicating whether a rule was broken.

### Particles
This class will allow the implementation of special effects such as fire, explosions etc. Since the particles need to be draw in 3D space correctly we need to use billboard technique. For pseudo code please look in the LEVEL DETAILS.

### Skybox
The skybox class represents the skybox. Who'da thunk? The constructor passes in the filename of the image to be used for the skybox. After the object has been created, the draw function draws the skybox.

## Terrain (optional)

For the terrain class the user need to pass in a bitmap which is not larger than 720 x 720 after the object the terrain we be created in the world space in 0, 0, 0, at last the user need call the draw function which draws each vertices on the screen, and for the terrain class the user also we also have a get bucket function which allows the user to get the current bucket which the player is standing on. the following pseudo code we represent how to make up the terrain class

## Sound

For the sound we will make two classes which is BGM(background music) class and Sound Effect class, each of which are given file paths for the audio to be used. After the objects have been created they will allow play, pause and stop functions in the class. Will also allow for looping of music. For details please look in the SOUND REQUIREMENTS

## Model

The model class we allow the programmer the load a model and move it more easily the code behind this class is Updating the matrix and send all the information to the video card and also let the programmer to access the each Mesh part in the mesh model. For details please look in the ART REQUIREMENTS

# 2 MENU DETAILS

## Buttons

For the Menu we will have the following buttons:

- Start
- Setting
- High Scores
- Exit

Start: When the user click the start button the user will allow to start a new game. The start button we be made in Photoshop for the design details please look at **DESIGN DOCUMENT**.

Setting: We will have a setting function when the user click the button and in the user function the user we be able the turn off the music and able to not use shadow in the game the following setting be save in a xml file as show below afterwards the user we have a back function that's allow the player to back to the main menu. the start button we be made in Photoshop for the design details please look at Design Document.

Exit:the exit function will close the application and clean up the objects that we created in the game or menu state after the objects has been clean the application we close immediately. The start button we be made in Photoshop for the design details please look at Design Document.

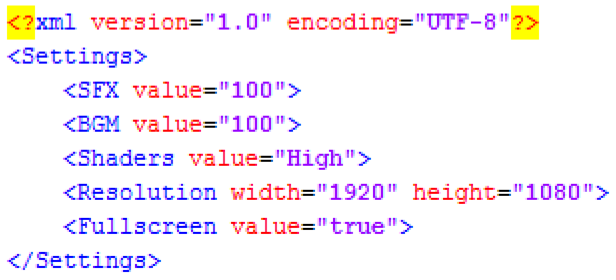
High Scores: Displays the top 10 scores. Defaults will be included when the game is first started; these will be overwritten as they are beaten.

## XML Layout format

```
<?xml version="1.0" encoding="UTF-8"?>
<LevelData>
    <Terrain File="Content/HeightMap.png" SpacingScale="1.0"/>
    <SceneryObjects>
        <Pencil X="10" Y="2" Z="23" Rotation="90" Scale="2.0"/>
        <Block  X="36" Y="2" Z="44" Rotation="45" Scale="1.2"/>
    </SceneryObjects>
</LevelData>
```

Each level in game has an xml file describing all objects and data for said level. The Level Data tag encapsulates all other tags or data for the level. The first tag should be for Terrain, which has two attributes:
**File**: The directory of the Height Map for Terrain, which starts in the same folder as the games executable.

**Spacing Scale**: The scale of the space between vertices in the terrain. Default is 1.
The next tag should be Scenery Objects, which encapsulates all game objects in the level. All tags within this should be named by what object they represent, followed by the following attributes:
**X**: The X position of the object.
**Y**: The Y position of the object.
**Z**: The Z position of the object.
**Rotation**: Rotation in degrees around the Y axis.
**Scale**: Scale of the model, default is 1.0



```xml
<?xml version="1.0" encoding="UTF-8"?>
<Settings>
    <SFX value="100">
    <BGM value="100">
    <Shaders value="High">
    <Resolution width="1920" height="1080">
    <Fullscreen value="true">
</Settings>
```

The settings for the game are stored in the layout above. The Settings tag encapsulates all of the settings types. The SFX and BGM (Sound Effects and Background Music) tags should have the following attribute:

**Value**: A value within the range of 0 - 100, which is the volume of the sound type.

The Shaders tag should have one attribute, which has three states: High, Low and None. High uses detailed shaders, Low uses the lower quality shaders, and None uses no shaders.

The Resolution tag tells the game the width and height (in pixels) of the window. It uses the following attributes:

**Width**: The width of the game window
**Height**: The height of the game window

The Full screen tag tells the game if the window should be full screen or not, and uses its attribute as a Boolean.

## Pseudo code in menu state class

////////////////////////////////////////////////////

**Update Function** →

```
////////////////////////////////////////////////////
if(Menu)
if(start click)//aabb
        //Start the game
if(Options click)//aabb
        //Go to options function
if(Exit click)//aabb
        //Exit the application
if(Options)
```

////////////////////////////////////////////////////

**Draw Function** →

```
/////////////////////////////////////////////////////////////////
if(Menu)
      //Draw all the buttons and model
if(Options)
        //Draw all the buttons and model
```

////////////////////////////////////////////////////

**Options Function** →

```
/////////////////////////////////////////////////////////////////
if(Shadow click)
      turn off / on the shadow

if(sound click)
      turn off / on the shadow

if(back Click)
      //switch back to menu
```

The **SOUND** and the **MODELS** we be represent in the **SOUND REQUIREMENTS** and **ART REQUIREMENTS**

# 3 LEVEL DETAILS

## Game Flow:

At the starting the Game state will go into the update functions and update the player(collisions, position , scale ,rotation). At this point the camera retrieves the position of the player and stores it. The NPCs are then updated (collision, position ,scale ,rotation), and finally update all the GUI (health , rules and speed). In the draw function we would need to draw all the level objects using the camera position as reference, then draw the GUI. The following list we show how to update in update function and draw function in the game state

```
Public void Update()
{
        Player.update();
        a_ocam.update();
        for each(NPC in NPCs)
                NPC.update()
        GUI.update()
}
Public void Draw(Camera a_ocam)
{
        DrawModels(a_ocam)
        DrawGUI()
}
```

## GUI

For the GUI in the game we separate into 3 parts

- Heath
- Rules
- Speed

The Health we decrease when the collision box has been hit if the collision box has been hit the health we decrease until it is 0 the collision box is still in the collision are the health we not decrease until the it goes away

The Rules we show up base on the rule base system // Paul

## Collision

**Wall and player:**

Collisions will always be defined between two entities (so, Scenery, Player or NPC subclass objects). Calculating if a collision has occurred will be done by taking the bounding boxes of the two entities and seeing if they intersect.

Collision response will be a reflection of the velocities of the two entities involved in the collision.

The Speed in the game we calculate by ?????

## Particles

For the particles system we would need to simulate the Fire explosion, smoke…..etc. the following pseudo code that how the Particles we be made

Update function:

1. Moving the Position
2. Calculate The Matrix
3. Change The Colour
4. Decrease the lifetime
5. If lifetime smaller than 0 set particle dead
6. Reset the particle to the default value

Draw Function

1. Check is the current particles dead(If dead do not do anything)
2. Enable Transplant
3. Use "Particles" Techniques
4. For the "Particles" Techniques we need to use the MVP and the Texture
5. Bind Vertex Buffer
6. Draw it

## Camera

To setup a camera is 3D space is different from we would need to setup a Matrix.LookAt and pass in the current position look at position and also the up vector in 3d space in camera class and also we would need to setup a Matrix. View and pass in AOF, Aspect Ratio and near ,far. And once to two matrix has been multiple it will give us the perspective matrix which is represent by below

Perspective = LookAt * View;

- **sics engine**

## Rules

Active rules are displayed in a stack (like, a visual stack) on the right side of the screen. Each rule will have its own icon, and display any relevant data (the NPC class it is relative to, the time left to fulfil a rule etc). New rules will be displayed on top of the stack. Rules that are broken will be removed from the stack.

Speed

Speed will be represented by a odometer on the screen. All the logic required will be a simple rotation of the needle, taking in the player's relative speed.

## Characters

Following is format to represent the charters

### Vehicles:

### ✧ Player:

- **Control Input**

### ✧ NPC:

- **AI state**

## Vehicles art assets:

The Vehicles we be find either online or provide by our artist. And the format will be in .x or .fbx. For the details of class setup please go to chapter COMPONENT INTERACTION AND INTERFACES.

### ✧ Player

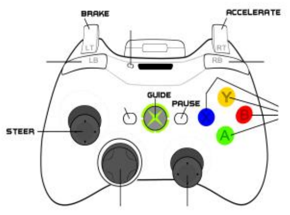
- **Control Input**

For the game we use using the following keys to play the game:

**PC:**

- W: Forward
- S: Backward
- D: Turn Right
- A: Turn Left
- Space: Break

**XBOX360:**



### ✧ NPC

Each NPC subclass will share a common base AI that runs when not in berserk mode. The pseudo code for this AI is as follows:

State has states NEUTRAL and SEEKING

```
if(State ==SEEKING)
{
    Seek()
}
else
{
    if(Near other vehicle)
    {
        State = SEEKING
        Seek(vehicle.position)
    }
    else
    {
        if(Near scenery)
```

```
                {
                        Avoid(scenery)
                }
                else
                {
                        Wander()
                }
        }
}
```

Methods used:
Seek()

This uses an implementation of A* to generate a path to a position on the grid, using the spatial index used for spatial partitioning. When State first moves to SEEKING, a position is passed into the method and a path is generated, but only partway. Let's say the path generated is five grid points long, but this may change for sake of performance and balance.

Once the path has been generated, the vehicle moves along the path each time Seek() is called. When the end of the path is reached, State is set to NEUTRAL .

Avoid()

The vehicle will attempt to drive around the position given.
http://gamedev.tutsplus.com/tutorials/implementation/understanding-steering-behaviors-collision-avoidance/

Wander()

The vehicle wanders around the level using a basic wander algorithm.
http://gamedev.tutsplus.com/tutorials/implementation/understanding-steering-behaviors-wander/

## Berserk AI

Katherine

Will continue in the same direction and reflect off of collisions.

Swarmer

Will use the Seek method as above.

Splitter

When the splitting occurs, the two created vehicles will move in new directions. The former will move in a direction 45 degrees to the left, the latter 45 degrees to the right. This avoids collisions between the two newly created vehicles.

Cloaker

Will use the base AI, will decrease its model's transparency as rules are broken.

## Terrain

```
#Load height map texture.
#Iterate through each pixel in texture and set height Data according to its color.
#Iterate through Terrain Width and height in nested for loop:
        #Calculate its X and Z position,
        # Set vertex position to X and Z

#Iterate through Terrain Width and height in nested for loop:
        #Calculate indices of four corners of a square in the vertex array
        #Add six indices for corners of triangles in the square to the index array

#Iterate through each IBO and calculate normal:
```

The **SOUND** and the **MODELS** we be represent in the **SOUND REQUIREMENTS** and **ART REQUIREMENTS**

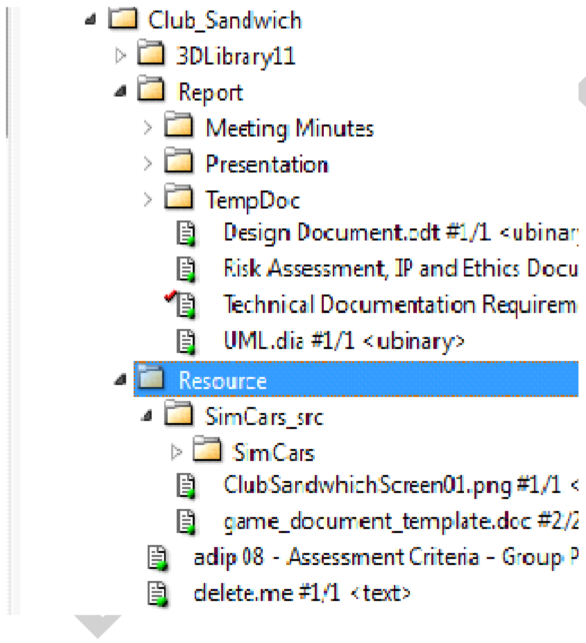
# 4 ENVIRONMENT DETAILS

## Perforce:

Perforce is software that allow team members is submit that work that they do and perforce will have a complete record that what we have done so we could go back any version that we could look or change

## Perforce Folder Layout:

The CLUB SANDWICH folder in perforce will be layout by the following format:
All the Report (Presentation, Meeting Minutes) will all be in the Report folder
All the Resource also needs to be keep track for our report and it will be in the Resource folder
The format is design at 10.17.2013



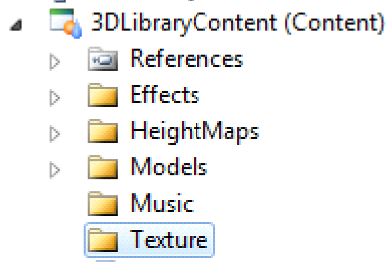
```
Club_Sandwich
  3DLibrary11
  Report
    Meeting Minutes
    Presentation
    TempDoc
      Design Document.odt #1/1 <ubinar
      Risk Assessment, IP and Ethics Docu
      Technical Documentation Requirem
      UML.dia #1/1 <ubinary>
  Resource
    SimCars_src
      SimCars
      ClubSandwhichScreen01.png #1/1 <
      game_document_template.doc #2/2
    adip 08 - Assessment Criteria - Group P
    delete.me #1/1 <text>
```

## XNA :

As out project we decide to use XNA framework XNA is provided by Microsoft that that facilitates video game development and management XNA is based on the .NET Framework, with versions that run on Windows, Windows Phone and the Xbox, the first release of XNA Game Studio, was intended for students, hobbyists, and independent game developers, XNA is a License free for window game XNA has publish more than 4 version and we are using the 4th version our group decide to use XNA Framework is because XNA Framework is easy to use and learn and it support cross-platform easily. The disadvantage of XNA framework is if we need to put our game on XBOX 360 we need create a lot of account first and also XNA framework will not its physics engine so if we need to do any physics we need to build our self or use other library.

## XNA Folder Layout:

The CLUB SANDWICH XNA Folder will be layout in the following format:



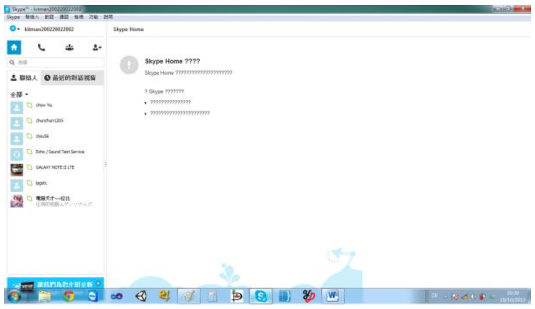
```
3DLibraryContent (Content)
  References
  Effects
  HeightMaps
  Models
  Music
  Texture
```

The format is design at 10.17.2013

## Skype:

Skype is a freeware software that allow people to communicate, transfer folder , send message and video chat with another people ,for our team we would need to use this software when we out of the school to communicate to each other also Skype sound quality could exceed normal phone so our team decide to use Skype

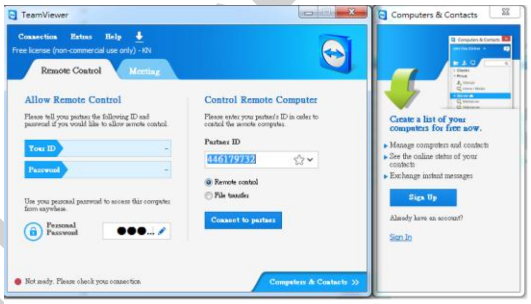
Interface of Skype



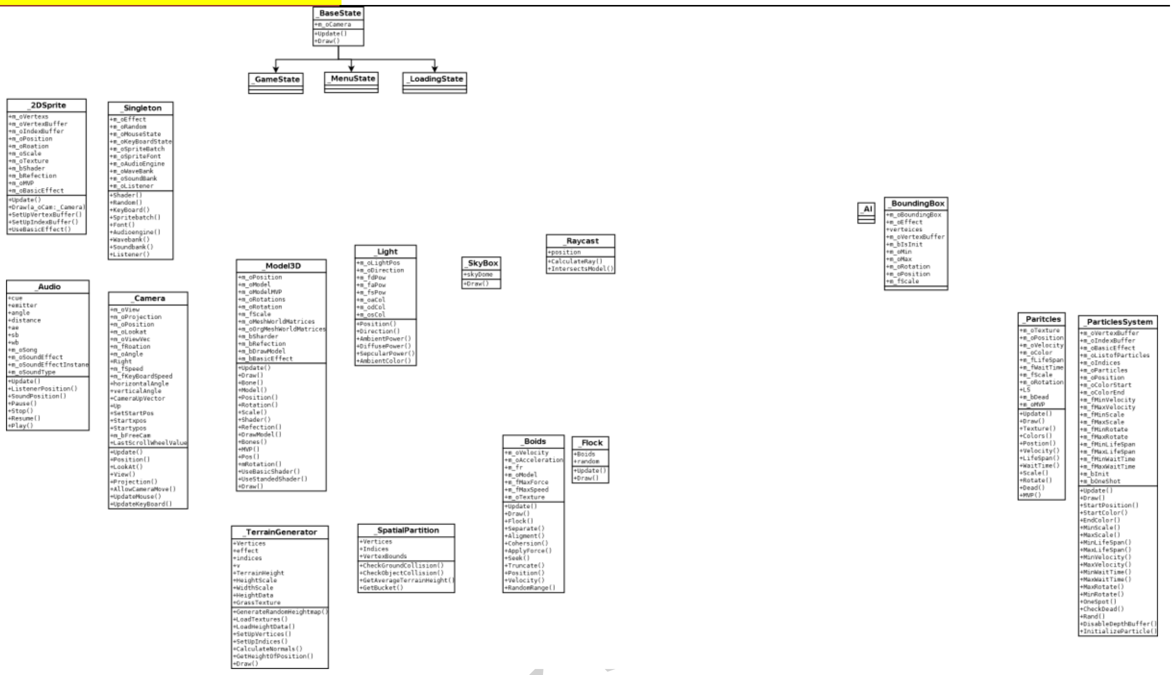
## Team Viewer:

Team viewer is a software that allows people control other people computer by provide the password the team viewer is easy to use If the one person need to control the password the another person just need to tell the person that what to control the computer the password and the ID , It is also a freeware Interface.The software also support other platform such as MAC, Linux, Android, Window Phone

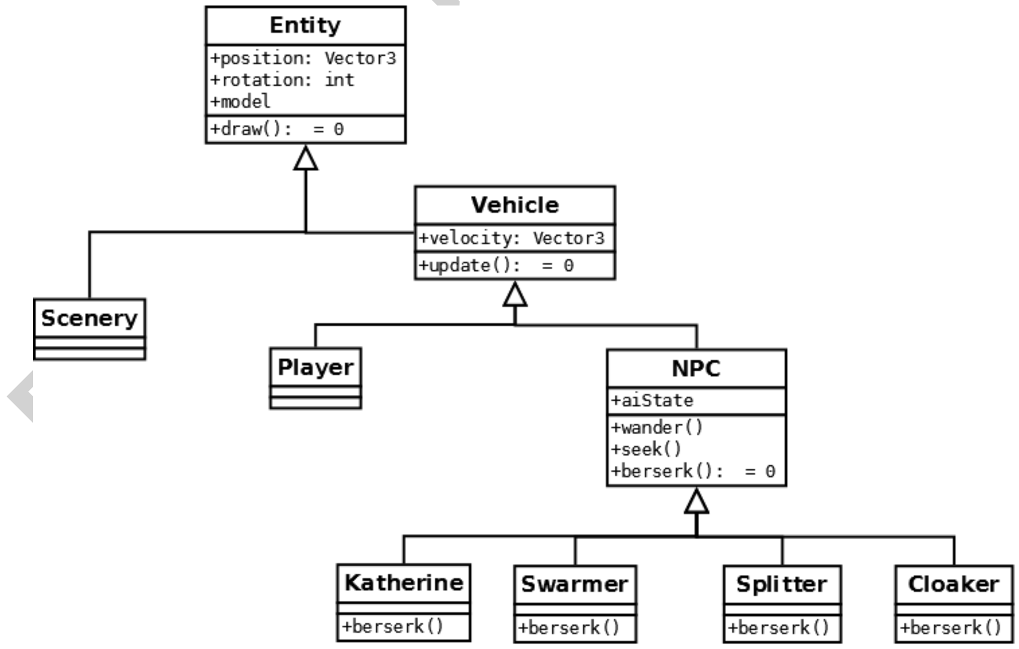
Interface of Team Viewer


TeamViewer
Connection
Extras
Help
Free license (non-commercial use only) - KN
Remote Control
Meeting
Allow Remote Control
Please tell your partner the following ID and password if you would like to allow remote control.
Your ID
Password
Use your personal password to access this computer from anywhere.
Personal Password
Control Remote Computer
Please enter your partner's ID in order to control the remote computer.
Partner ID
446179732
Remote control
File transfer
Connect to partner
Not ready. Please check your connection
Computers & Contacts
Computers & Contacts
Create a list of your computers for free now.
Manage computers and contacts
See the online status of your contacts
Exchange instant messages
Sign Up
Already have an account?
Sign In

# 5 COMPONENT INTERACTION AND INTERFACES



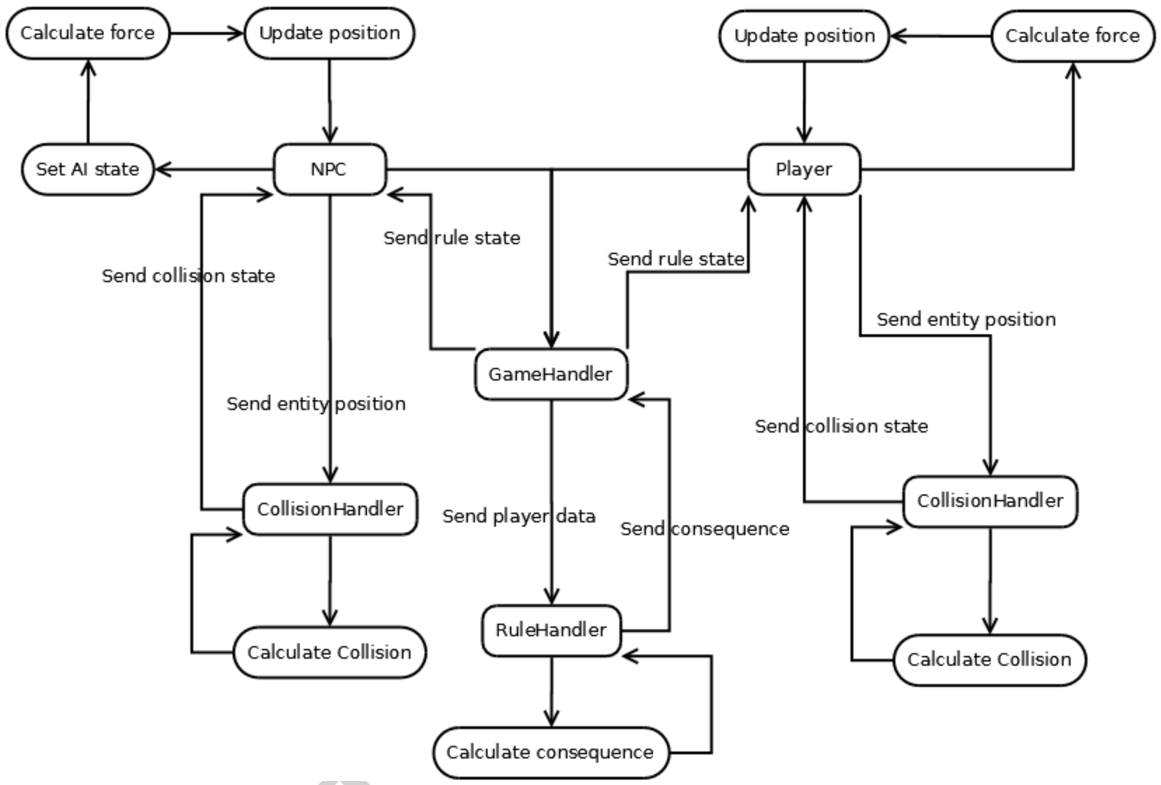
## Entity Breakdown



Every object in the level is classified as an Entity, then defined from there through inheritance.

-Scenery refers to the non-active level elements; walls and such.
-Vehicles are self-explanatory. There will only ever be one player, but there will be multiple enemies as NPCs. The subclasses of NPC refer to the enemy classes. Each have their own implementation of the berserk AI state.

## Data Flow Diagram



This diagram shows the basic game loop for player interaction with the level and the enemies.
-Collisions are handled between the given Vehicle and all entities within the same partition by the Collision Handler, the result of which is returned back to Vehicle. The other entity positions are retrieved through use of a singleton pattern.
-Player data sent to the Rule Handler would be data relevant to the active rule definitions. Almost all of the rules can only be broken when the player has destroyed an enemy, hence sending the player's last destroyed enemy type would be sent. Other rules relate to the player's speed, in which case the player's velocity would be sent. The rule handler then checks the sent data against the active rules and lets the Game Handler know if a rule has been broken.
-Data flow to the Player and NPC objects are much the same, except Player receives control input and NPC receives any changes in rule state. Control input will change the force calculated in player, and the rule state will affect the AI controlling the NPC.

# 6 ART REQUIREMENTS

## Limitations:

- For XBOX Model Format : FBX ,.X , OBJ
- For Window Model Format : FBX ,.X , OBJ

## Menu

For the Menu for the environment will would need the following Object/sprite

### GUI

- 3 Sprites(Start Button , Options Button , Exit Button)
- 3 Text(Start Text, Options Text, Exit Text)
- 1 tick box

### Game Objects

- 1 lighting
- 1 race model(animation)

## Game

For the Menu for the environment will would need the following Object/sprite

### Characters

- 1 Player model
- 4 enemy

### Game Objects

- 1 house
- terrain(optional)

### Game play Elements

- 3 Sprites(Speed col, Rules , heath )

## Resource

The resource we be either provide by our artist or on the Internet. For the details of the Art design please look at **DESIGN DOCUMENT.**

# 7 SOUND REQUIREMENTS

## Limitations:

- For XBOX Sound Format :WAV , MP3
- For Window Sound Format :WAV ,MP3

## Menu

- 1 Background Music(We be provide by artist)

## Game

- 1 Game Explosion
- 1 Car crash
- 1 Background Music(We be provide by artist)
- 1 Car Moving Sound
-

## Resource

The resource we be either provide by our artist or on the Internet.

# 8 HARDWARE REQUIREMENTS

**XBOX360**

- **Putting a game into XBOX360**

## Hardware Requirements

**Custom IBM PowerPC-based CPU**
Three symmetrical cores running at 3.2 GHz each
Two hardware threads per core; six hardware threads total
VMX-128 vector unit per core; three total
128 VMX-128 registers per hardware thread
1 MB L2 cache
**CPU Game Math Performance**
9.6 billion Dot product operations per second
Custom ATI Graphics Processor
1 0 MB of embedded DRAM
48-way parallel floating-point dynamically scheduled
**Shader pipelines**
Unified shader architecture
**Polygon Performance**
500 million triangles per second
Pixel Fill Rate
16 gig samples per second fill rate using 4x MSAA
**Shader Performance**
48 billion shader operations per second
**Memory**
512 MB of 700 MHz GDDR3 RAM
Unified memory architecture
**Memory Bandwidth**
22.4 GB/s memory interface bus bandwidth
256 GB/s memory bandwidth to EDRAM
21.6 GB/s front-side bus
**Overall System Floating-Point Performance**
1 teraflop
**Storage**
Detachable and upgradeable 20-GB hard drive
12x dual-layer DVD-ROM
Memory Unit support starting at 64 MB
**I/O**
Support for up to four wireless game controllers
Three USB 2.0 ports

Two memory unit slots

## Limitations of the Xbox 360

The Xbox360 is an aging console, with hardware specs much lower than the PCs used by the average gamer of today. The limitations include:

- 512MB of memory which is shared between the CPU and GPU.
- Save game data has a maximum size of 52MB
- The maximum size of an XNA project is 2GB

# 9 Time Line and Milestones

For the time line and milestones please look at timetable.xls.