Programming Lead: Nathan Contreras

[company name tentative]

Technical Director:  Patricia Sipes

Design Direction: Juan Carlos Gozalez

Maya game \*

\* Title Tentative

* Target Hardware/Platform:
  + PC
    - Windows OS, Mac OSX, Linux OS
* Development Environment/Tools:
  + Unity 5.3.4f1
    - Level design engine
  + Visual Studios
    - Scripting environment
  + 3DS Max/Autodesk Maya
    - Modelling/Rigging environment
  + Photoshop
    - Texturing environment
* Workstations Used:
  + Personal Laptops
    - Various operating systems and performances
  + School Workstations
    - Windows 7 OS
    - Intel Xeon CPU 3.50 GHz (12 CPUs)
    - 16384MB RAM
    - NVIDIA Quadro K4000 video card
      * 4095 MB Video RAM
    - Dual 1920x1080 (32-bit) resolution display
* [DEVELOPMENT PROCEDURES AND STANDARDS]
* Coding Conventions
  + Camel Case (examples follow)
    - playerDirection
    - isPlayerGrounded
    - weaponArr[]
  + Booleans as questions
    - isPlayerGrounded
    - isPlayerAlive
    - hasPlayerStopped
    - hasEnemyDied
  + no spaces in file names
    - Good:
      * “HandyDandySprite.png”
    - Bad:
      * “Handy Dandy Sprite.png”
  + Variable and method names should avoid using underscores
    - Good:
      * initBullet();
      * mySpriteObject
    - Bad:
      * init\_Bullet();
      * my\_Sprite\_Object
  + Member variables should be prefixed with “my”, avoiding the use of the prefix “m” where possible.
    - Good:
      * mySpriteObject
      * myBulletObject
      * myAnimator
    - Bad:
      * mSpriteObject
      * mBullet
      * mAnimator
  + Comments should be aligned above the line they refer to. If impossible, align to the right.
  + Variables should be named appropriately.
    - Good:
      * int enemyCount;
      * float deltaTime;
      * Vector3 myPos;
      * Enemy\* ptrEnemy;
    - Bad:
      * char\* ptr;
      * string\* ptr;
      * float b;
      * int a;
  + Methods/functions should be named appropriately.
    - Good:
      * int add(int first, int second);
      * Vector3 calcDirection(Vector3 startPos, Vector3 endPos);
    - Bad:
      * void a(float b, float c);
      * char\* strstr(string a, string b);
  + Have some form of comment above your function/method that describes its function, return, and formal parameters.
  + Methods/functions are to be separated with the following comment, adjust length as necessary
    - //==============================
  + ALWAYS have curly braces around if/else, while, for, do while, and other loops. This is a very safe practice and can prevent unnecessary errors from occurring.
* Scripting Language
  + C#
* Directory Structure
  + trunk\
    - Documents\
    - GameSimFall2016\
      * Assets\
        + Animations\
        + Audio\
        + Materials\
        + Models\
        + Prefabs\
        + Scenes\
        + Scripts\
        + Textures\
      * ProjectSettings\
    - Sandbox\
* Version controls and tools
  + GitHub hosted repository
  + Accessed via TortoiseSVN
* Game Engine
  + Unity 5.3.4f1
* Scripts List (Tentative)
  + CameraController.cs
  + Game.cs
  + Movement.cs
  + Rotation.cs
* Graphics file requirements
  + File type: .PNG
  + size: tentative
  + format: tentative
  + naming convention:
    - Good:
      * “HandyDandySprite.png”
    - Bad:
      * “Handy Dandy Sprite.png”
* Audio file requirements
  + File type: mp3
  + size: tentative
  + format: tentative
* Pipeline
  + **Description** – A rough description of how the script will overcome the task at hand.
  + **Functional Decomposition** – Decompose the problem into smaller, simpler problems.
  + **Pseudocode** – Rough outline of what the final code will be expected to perform.
  + **Adding script to Repository.**
  + **Coding** – Begin translating pseudocode into the C# scripting language, attempting to make a working script.
  + **Commit changes to the script to the Repository**.
  + **Testing I** – Begin using the created script, testing for errors and bugs, making note of edge cases.
  + **Debugging** – Fixing the errors and bugs found in the script.
  + **Testing II** – Test the changes made to the script, return to *debugging* stage if more errors/bugs occur.
  + **Commit changes to the script to the Repository**.
  + **Revision** – Cleaning up any inefficient/spaghetti code into refined/functional code snipets.
  + **Testing III –** Test the revisions made to the script.
  + **Debugging II** – Fixing any errors/bugs made during revision.
  + **Testing IV –** Test fixes to the revised script. Return to Debugging II if more errors/bugs occur.
  + **Documentation** – Adding missing comments/documentation to any code blocks.
  + **Commit changes to the script to the Repository**.