Advanced Diploma of Professional Games Development

Gravitas

Technical Documentation

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

Document Information

Table Of Contents

Document Authors

Author	Position
Andrew Giannopoulos	Son #2 (the inferior)
Iain Dowling	Son #1 (the superior)
Jackson Luff	Father, the bearded woman

Advanced Diploma of Professional Games Development

Gravitas

Technical Documentation

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

Related Documents

- Design Document: https://docs.google.com/document/d/12blFs2Voaw727ChL6irqyyu9kyujHX4GF19FH9dJhPQ/edit
- Project scope: https://docs.google.com/document/d/14uxLQ4Q3Ce0qj4vZ6HgpkNlcoh26Cw5NFqnVjFSD5hs/edit#
- Coding standards: https://docs.google.com/document/d/1735Hn36S8tlDKGpDuErt5rEdNCwREC51UdY9wwQ-aMM/edit

1: Introduction

Gravitas is a single-player 2D puzzle-platformer in which the player must use their ability to manipulate gravity to overcome a series of challenges. The game is to be available on the PlayStation Vita and Windows PC platforms, and will feature an abstract art style of glowing brightly coloured angular polygons and enemies. Gameplay will be initially slow-paced as the player becomes familiar with the controls, but will gradually ramp up in both difficulty and complexity as they progress through the game.

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

2: Platforms

2.a: Target Platforms

The target platforms will be the PlayStation Vita and Windows PC.

2.b: Technical Specifications

Vita Specifications:

CPU: ARM® Cortex™- A9 core (4 core)
GPU: SGX543MP4+
SCREEN:
5 inches (16:9), 960 x 544, Approx. 16 million colours, OLED
Multi touch screen (capacitive type)
SOUNDS:
Built-in stereo speakers
Built-in microphone
RAM:
512 MB
(128 MB VRAM)
MEMORY REQUIRED:
35 MB

PC Specifications:

CPU: Intel Core i3 4150 (or above)
GPU: integrated graphics (or above)
SCREEN:
944px x 544px LED/LCD Display
SOUNDS:
External speakers
External microphone
RAM:
1GB Dedicated (Min)
MEMORY REQUIRED:
512 MB

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

2.c: Build Process

PC Version:
The version of Gravitas that will be built for the PC version, running on Microsoft Windows, will make use of InstallSimple. InstallSimple is a free, simple installation package maker that will allow us to create an installer for Gravitas. Using the files we provide it, InstallSimple will create an installer that can be distributed. The installer comes in a wizard format that will walk the user through the installation process, show them the licenses for the game and allow them to chose their installation location for Gravitas' files.

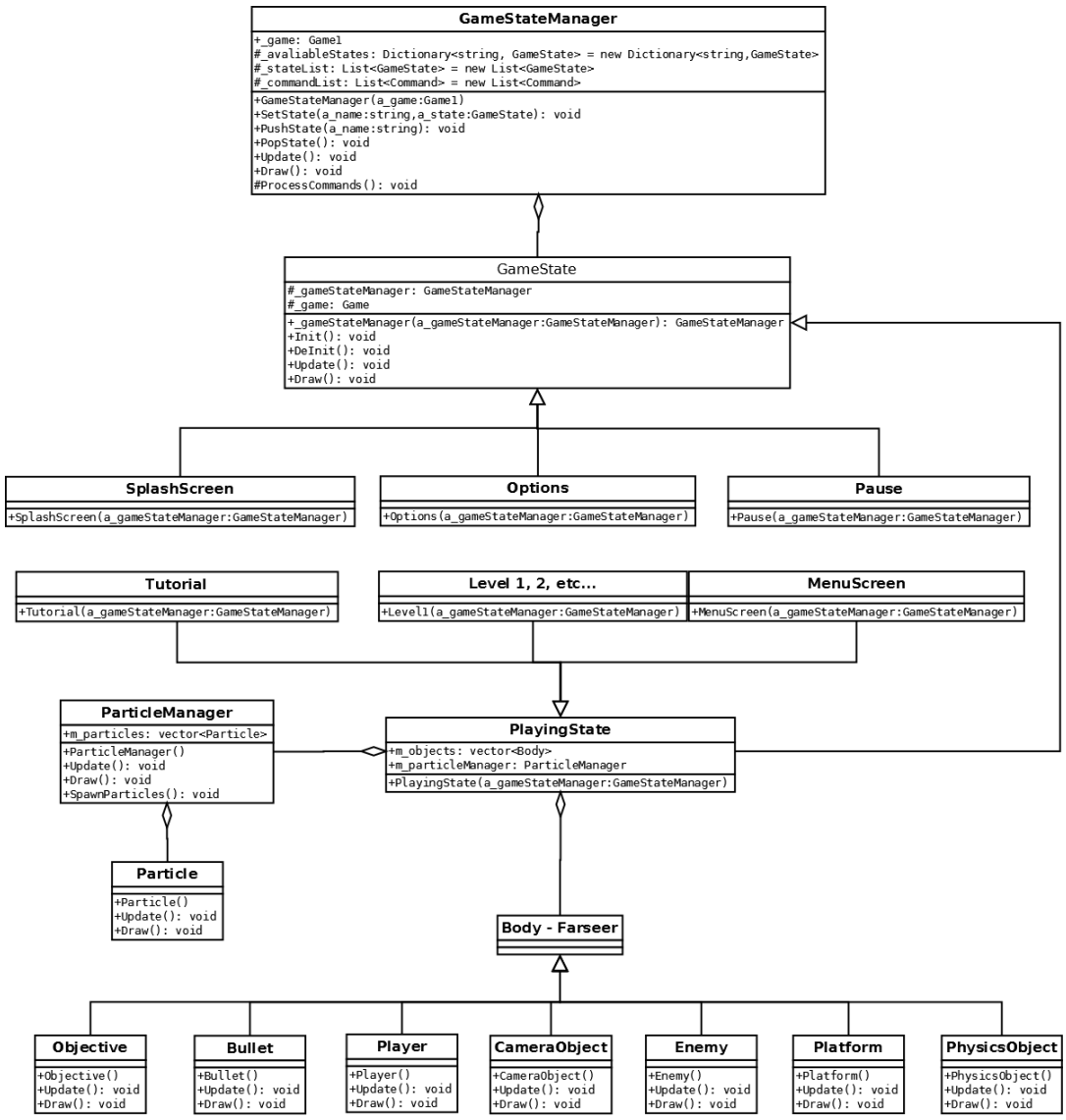
PS Vita Version:
Gravitas for PS Vita will use the Content Manager Assistant for PlayStation. The content assistant manager allows the transfer and backup of files between the PS Vita and the PC, and runs on both Windows and Mac. Using this program, the user will be able to transfer Gravitas for PS Vita files from their computer to be installed in the Vita.

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

3: Code Specification

3.a: UML

Class diagram:



Academy of Interactive Entertainment

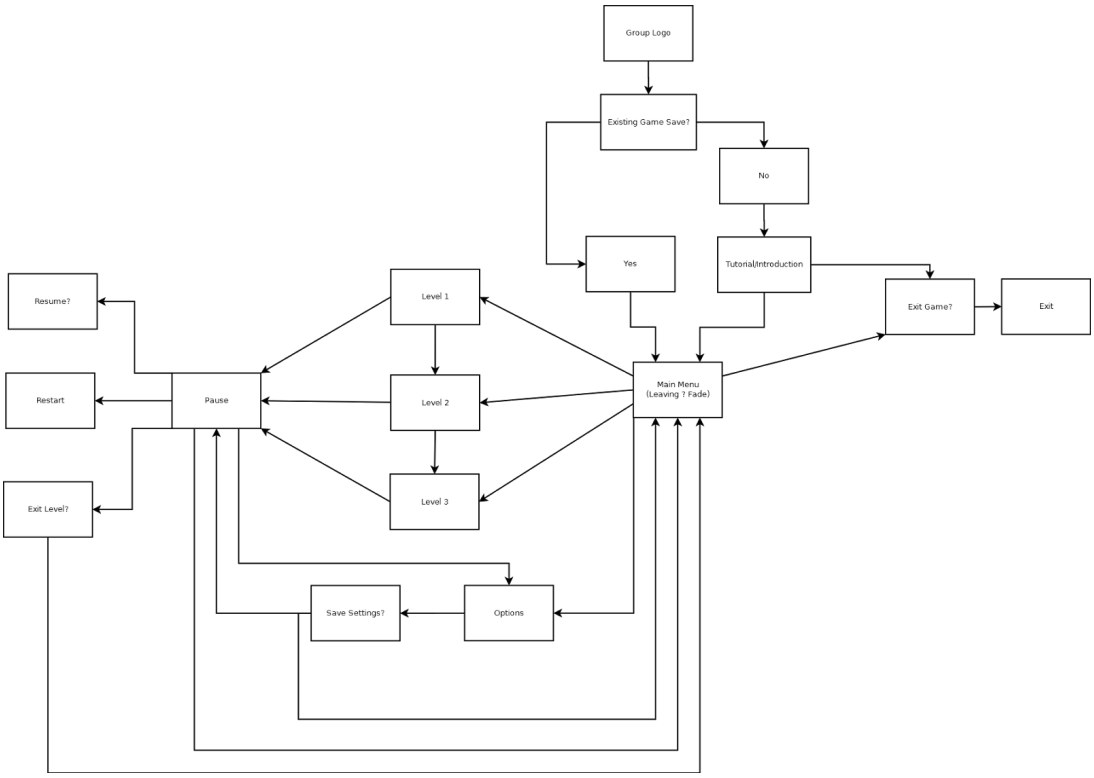
Advanced Diploma of Professional Games Development

Gravitas

Technical Documentation

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

Game States flow chart:



Loading/Unloading assets:

- If a game save happens to exist, ignore tutorial.
- From level to level, a start and end loading / unloading procedure is applied - to provide fluency (not to mention efficient on memory instead of unloading in run time)
- If entity dies; unload. (As well as enemy bullet restriction)
- Levels are loaded in bulk (but in theory are chunks put together as whole)

Academy of Interactive Entertainment

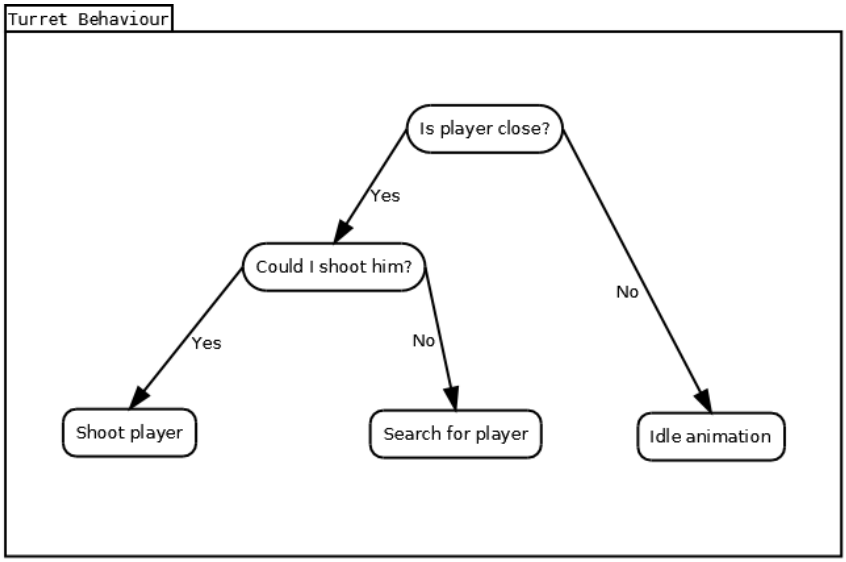
Advanced Diploma of Professional Games Development

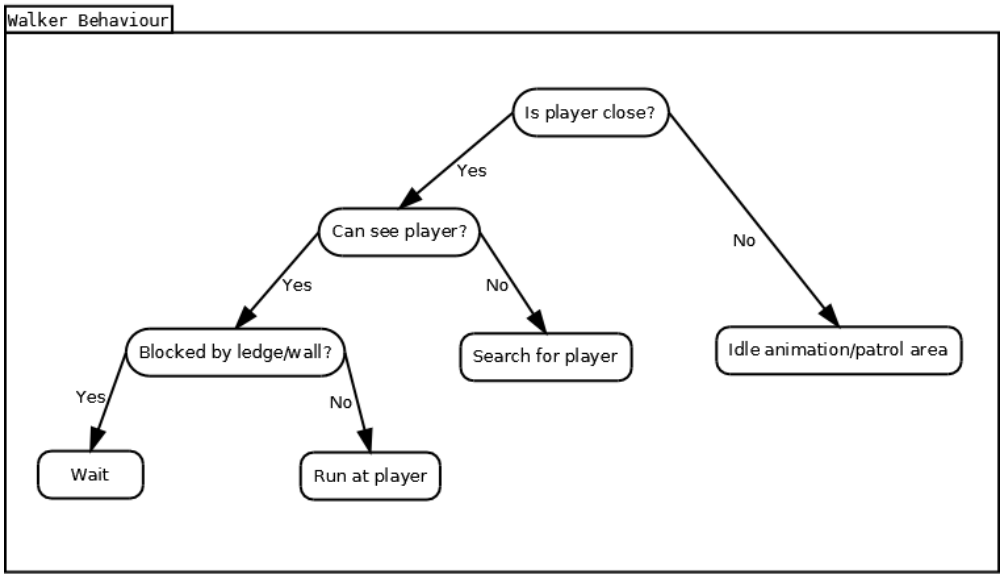
Gravitas

Technical Documentation

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

Game Entities:





- The enemy entities will have a reaction radius around them. As soon as the player enters their radius - they then begin to attack.
- Collision detection for all entities is ran through Farseer physics engine
- The entities that intake a body essentially keeps track of world position. It is basically a point is space that is affected by forces such as impulses from collisions and gravity.

Gravitas

Technical Documentation

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

GUI:

- There will be very little in the way of GUI that overlays the gameplay.
- There will likely only be one energy bar that represents how much you can change gravity while in the air
- It will be called your **"gravitational potential energy"**
- Main and Pause menus will be more akin to small "levels" in that you still control your player and can interact with gravity in much the same way as you do in-game, however the paths you take determine which options you select or where you want to go.
- The gravitational potential energy bar will usually be about 50% transparent as a white bar, except while the user is holding down the right-trigger at which point it will become completely opaque and gain a white glow around it to highlight your use of it.

Save Games:

- Save games will be saved in .bin files that are named for the profile names
  eg. "Andrew.bin" will hold the save game data for the profile named "Andrew".
- Users will be able to select a profile if one exists upon starting the game (directly after the splash screen)
- If a profile does not exist, the player will go straight into the tutorial from the splash screen.
- The profile contains data on the latest level and checkpoint the player has reached.

3.b:Third Party Libraries

- Monogame: http://www.monogame.net/
- Farseer Physics Engine: https://farseerphysics.codeplex.com/

3.c:Tools

- MSPaint was used for custom editing / design
- paint.NET for level design and image manipulation
- Photoshop for character / Object creation
- Visual Studio 2012 for code production
- Google docs for documentation
- Dia for UML/Flow chart visualisation
- Trello for organisation
- Audacity for sound manipulation

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

3.d:Coding Standards

https://docs.google.com/document/d/1735Hn36S8tlDKGpDuErt5rEdNCwREC51UdY9wwQ-aMM/edit

4: Assets Specifications

4.a: Asset formats

Textures:
All assets coming into our game will be in the PNG format so they are all of the same quality and to help with consistency. The limitations applied purely from Monogame are as follows; 2048 for Reach and 4096 for HiDef. Not only should we keep this in mind, but also that the game will be running on Vita also, and thus we must conserve memory usage via high resolution textures.

Animations:
Our animations will be kept in a spritesheet, one large PNG containing all the frames of all the animations (jumping, walking, running) for one object (player, enemy, etc) that will be loaded into the game at the beginning of each level. There will be multiple spritesheets loaded into the game. the monogame rectangle class will be used for for holding the frame measurements displaying the current frame in the spritesheet. This will be done by having an incremented variable multiplied by the width of the texture. The actions will be held in a listing format where for every action is a new marking on the Y axis for the frames height of the texture.

Sounds:
Sounds files will be stored as a .wav. Wav files deliver high quality audio frequency waves and although the file consumption rate is more intense Wav is currently the only sound format that works on both Monogame and PSM and thus provides us general capabilities (no console specific code) whilst coding audio. Each bit depth will vary, but the sample rate will be 44100 .

Fonts:
For this project our default formatting font is *Arial*. This specific font is simplistic and easy to read. Thus, useful for all age groups. The font will be 30px large, non-bold, non-italicized and black.

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff

Level data:
Data will be saved off in binary format. The file will contain three integers. The first represents the most recent level the player has unlocked in the game. The second two will determine the level and checkpoint that the player was last at, respectively. No other level data will need to be saved, as enemies are planned to respawn upon loading a level.

4.b: Asset Pipeline

Textures:
1. Textures are created in photoshop and paint.NET.
2. The textures are then manipulated into multiple poses and saved into a separate PNG to create a sprite sheet.
3. The spritesheet is saved into the Content/art folder inside the project.
4. The spritesheet is loaded into the game and the Animation class separates each frame in the image to create animations.
5. The 'Animation' class will loop through the width of the texture (from left to right) over a frame that's masked over the spritesheet. The rectangle of the frame will move downward. (Incrementing via the frame height) Depending on the Y position, depends on the chosen action.

Sounds:
1. Sounds are found on free online resource websites.
2. They are downloaded in any format, then converted to .wav format using Audacity.
3. The sounds are then saved into either Content/Music or Content/Sound, depending on whether it is a sound or music asset.
4. Sounds and music are then loaded in when the game starts.

Fonts:
1. Run a .spriteFont file in an XNA Project to output a .xnb file into the bin
2. Collect the .xnb then include in the Content folder.
3. Use as you would a normal .spriteFont file.

Level Data:
1. In-game level editor to allow for on-the-fly creation/deletion of platforms/enemies
2. Level data is saved to .XML files
3. Level data can then be loaded in from this file when the game starts

4.c: Folder Structure and file naming conventions

Assets will be stored inside the Content folder in the project:

- Spritesheets will be located in Content/Art
- Sounds will be in Content/Sound
- Music will be in Content/Music
- Fonts will be in Content/Fonts

Textures and sprite sheets will be located together as they will be processed by the same class, the animation class, even if they have no animation to them.
Work-in-Progress files will be stored in '*WorkInProgress'*, a folder that will be kept separate from the Project files. The '*WorkInProgress'* folder will be located in the Perforce directory so that all team members can access the files and get the latest revisions. Files in this folder that are considered finished will be moved into the project files.

5: Save File Formats

5.a: User profiles

Save games will be stored in .bin files. The name of the file will be the player's profile name, and the file will contain three integers. The first will indicate which is the most recent level the player has unlocked in the game. The second two will determine the level and checkpoint that the player had last saved at.

5.b: Game data

Also stored in .bin files, game data that is saved will include number of deaths (rewritten to the file each time the player dies) as well as total playtime in the game.

5.c: Level data

Level data is saved in .xml documents, with pairs of integers representing the end points of platforms or locations of enemies in the game.

Academy of Interactive Entertainment

Advanced Diploma of Professional Games Development

Gravitas

Technical Documentation

Delta Force: Andrew Giannopoulos - Iain Dowling - Jackson Luff