



Evan, Fletcher, Jehremy, Scott, Simon

Technical Design Document



Table of Contents

Game Overview	5
Game Features – Technical.....	5
Light Field.....	5
Application.....	5
Effect.....	5
Duration	5
Limitation.....	5
Heavy Field.....	6
Application.....	6
Effects	6
Duration	6
Limitation.....	6
Bella	7
Application.....	7
Effects	7
Limitation.....	7
Movement	7
Technical Implementation	7
Achievements.....	8
Application.....	8
Technical Implementation	8
Telemetry System	9
Application.....	9
Technical Implementation	9
Camera	10
Overview.....	10
Camera Features.....	10
Cinematic/Cutscene	10
Technical Implementation Plan	10
The Game World.....	11
Overview.....	11
Technical Implementation Plan	11
Game Characters.....	12

Overview.....	12
Shader	12
Protagonist	12
Technical Implementation	12
Shader	12
Protagonist	12
Antagonist.....	12
Allies.....	12
Engine	12
Unreal Editor 3.....	13
Technical Requirements.....	13
Technical Risks.....	13
Custom Animations	13
Physics Solving.....	13
Additional Specs	13
Folder Structure	13
Naming Conventions	16
Testing.....	17
Overview.....	17
POC.....	18
Early Prototype	18
What we did:.....	18
What we planned to do:	18
Limitations / Success:	18

Game Overview

Curvature is a third person action game. You play as Bella, a teen aged girl. Chosen as the bearer of the Relative Field Device, a device capable of encapsulating her inside relative physics fields, making her either light as a feather, or heavy like a bowling ball. Her journey begins with the death of her mentor, the previous bearer, and continues as she uncovers a mysterious plot to over-throw the government. In the end, Bella must take up the task of saving her broken world.

Game Features – Technical

Light Field

Time Allocation (16hours)

Application

The Light field is applied by hitting the light field button; see Controls Section in the Design Document for further details. If the player is in an environment where the field cannot be deployed (small confined area) a negative sound will emit when the player hits the field select.

Effect

Applying the Light Field effect encapsulates Bella in the light field, making her feather weighted. The light field is easily affected by forces in the environment, such as air. It floats only slowly towards the ground, and will bounce off virtually anything without hurting it. It has a very low physics mass, and a brake is applied if a negative Z velocity is detected, allowing it to travel very quickly upwards while floating only slowly down. When applying the field it will take into account the current direction and velocity of Bella, continuing her on in the same direction and speed. The light field also has the ability to jump. A Boolean (bCanJump) is flagged true while a jump is valid to be triggered. This is based on two tests: The first is built into Unreal, the `bIsChassisTouchingGround`, reads if the field is currently in contact with the ground, the second test is a ray cast fired direct down the negative Z axis, which tests each tick for the distance from the center of the field to the ground. `bCanJump` is flagged true if `bIsChassisTouchingGround` is also true. If the distance to the ground is read to be above a certain threshold `bCanJump` is immediately flagged false, and when `bIsChassisTouchingGround` reads false a timer is started which will flag `bCanJump` false after .5 seconds. This provides a window for the player to be able to successfully jump even if the field is still bouncing slightly either due to uneven ground or a recent impact.

Duration

The effect once applied will last until the button is released, or the player hits the heavy field button to switch field types. The jump is available to the player whenever they are on the ground. See “Appendix: Physics Field Mechanic”

Limitation

- Not available in certain confined areas where there is insufficient room to spawn the mesh.

Heavy Field

Time Allocation (16hours)

Application

The Heavy field is applied by hitting the heavy field button. If the player is in an environment where the field cannot be deployed (small area) a negative sound will emit when the player hits the field select.

Effects

Applying the Heavy Field effect encapsulates Bella in the heavy field, making her extremely heavily weighted. The heavy field is resistant to most forces in the environment, such as air vents. It falls quickly towards the ground, and can crush or smash some objects in the environment (eg Energy Generators). It has a very high physics mass, and a built in momentum system to exaggerate the effect of the moment built into the physics engine. This is done by increasing the field's velocity by a percentage of the field's current velocity. To increase the heavy field's "heavy" feeling it does not build momentum upwards, and if the Z Velocity is above a certain threshold a negative Z force is applied to cause it to begin falling faster. When applying the field it will take into account the current direction and velocity of Bella, continuing her on in the same direction and speed. The Heavy field also has a speed boost. The boost applies a set force along the X and Y Axis. The amount of force (positive or negative) applied to each axis is based on a boost strength variable (tunable), and the direction of input from the left control stick relative to the camera's current rotation. The boost also applies a percentage of the current Z velocity, if it is less than 0, as a downward force. For example if the camera is pointing in the direction the field is travelling and the player is pressing up ("forward") on the Left Stick then the boost will apply in the direction that the player is travelling, if in the same situation the player were to press down (backwards) on the left stick, the boost would apply against the player's current momentum acting as a break. If the camera is rotated 90° to the left of the direction the player is currently travelling and the player's activates the boost while pressing up on the left stick, the boost will apply perpendicular to the player's current direction of travel causing the field to turn quickly.

Duration

The effect once applied will last until the button is released, the field is overloaded, the Device runs out of energy or the player hits the light field button to switch field types. The speed boost lasts 0.5 seconds and is available no more often than once every 2 seconds. See "Appendix: Physics Field Mechanic"

Limitation

- Not available in certain confined areas where there is insufficient room to spawn the
- mesh.

Bella

Time Allocation (16hours)

Application

The player is not in Bella form if the Heavy or the Light Fields are active. The player can always return to Bella form whenever they wish. The player will maintain momentum from their previous form when switching to Bella form.

Effects

Bella form acts as a normal Unreal Pawn, including health and a modified fall damage. Bella form is the only form in which the player is vulnerable to damage, and may die. It will have a very high air control, and a standard jump. Bella is also capable of moving much faster in the air than on the ground, this is necessary as the speeds the field travel in the air require her to have a much increased movement speed to be useful. However that level of movement is disconcerting to the player while running around on the ground and it becomes excessively difficult to maneuver with precision on the ground, one of the primary goals of the Bella Form. Bella form also has the ability to trigger a special speed boost when switching to the heavy field, and a jump boost when switching to the light field. This is trigger simply by switching to either of those fields after remaining in Bella form for at least 1 second. A Boolean is set in the PlayerController class to track if the player has been Bella for long enough to trigger the boost. That Boolean is set by a timer within the PlayerController class which is started as soon as the player returns to Bella form. The Jump boost applies a set force (JumpBoostStr variable, tune-able) along the positive Z Axis. The speed boost applies a percentage of the player's X and Y velocities (HorSpeedBoostStr variable, tune-able) and a lower percentage of the player's Z velocity (VertSpeedBoostStr variable, tune-able), as forces along their respective axis.

Limitation

- Not available in certain confined areas where there is insufficient room to spawn the mesh.

Movement

- Player can jump.
- One jump at a time (no double jump).
- Strafe is same speed as running
- Player face direction right joy-stick controlled.
- player can run and walk (analog based on Control Stick)

Technical Implementation

Implementation of the fields is based off of the native functionality of the Scavenger Vehicle in UT3, using its ability to roll and bounce. When the player enters, first their collision is disabled, then a modified Scavenger Vehicle is created and the player is placed inside it. The player then gains control of the field, instead of their pawn. Each Field Vehicle has the properties of both the light and heavy fields and is able to switch between the two at will. The player will remain inside this vehicle until they attempt to return to Bella form at which point the player is removed from the vehicle and the vehicle is destroyed.

Achievements

Time Allocation (16hours)

Application

We will build an achievement system to provide more challenging goals to more advanced players. There will be 22 Achievements in all, and they will be listed on a screen in the pause menu. The main pause menu will also show the last 3 achievements earned.

Technical Implementation

The Achievements are stored in the T6_Achievements class, this is created within the Game Replication Info, or GRI. This requires the Default GRI class to be changed to our own, T6_UTGameReplicationInfo. The class will contain an Boolean variable and a function containing any test conditions that need to be run, for each achievement. The class will also contain a function to trigger effects when achievements are unlocked. This includes the particle effect, sound cue, and updating the Achievements Menu and the last 3 achievements earned. The achievement will all be triggered by their respective test functions, which will be called within the game code passing appropriate test parameters are arguments to the functions, or by being triggered directly by one of the custom kismet nodes we have built for our level design team. The Last three achievements will be accomplished by creating a struct contain two strings, one for the Name of the achievement and one for the Description of the achievement. The a dynamic array of these is created and each achievement as it is unlocked is stored within the array. The array is limited to a size of 3 and uses a FIFO (first in first out) system to keep only the 3 newest entries. These entries are then read by a custom UI Kismet Node and used to populate the UI Panel which displays the last 3 achievements. A second array is kept containing a struct for each Achievement, holding its Name and Description, each achievement is assigned an index within that array, and that index is used as a argument passed to the achievement unlocked function to cause the update to happen at the same time the achievement is triggered.

Telemetry System

Time Allocation (16hours)

Application

We are going to create a telemetry/in-game player feedback system for use in Curvature. The system will store the player location, and state per tick, as well as events for achievement unlocked, player death, jump activate, boost activated, transition jump activated(Bella to Light), transition boost(Bella to Heavy) activated, and user inputs (X for Having Fun, B for Not Having Fun) for “Having Fun” and “Not Having Fun” bound to the controller.

Technical Implementation

The Unrealscript will log out the appropriate data into it's log file(player location, player state etc.) and we use a delimiter in front of the data to distinguish what type of data is on each line, ad a global delimiter to set our telemetry data apart from the rest of the log.(T6TELX for location and state, T6TELX A for achievement etc).

The log is then immediate parsed by a pearl script which sorts out only telemetry data and removes the rest and the T6TEL delimiter from the file. It also removes any lines in which the player's location does not change from the previous. That file is then passed to a MaxScript file inside Autodesk 3DSMax which creates a 3D representation outlining the player path, state, any any of the “events” which are marked with appropriate models along the player path. That model is then automatically outputted in the .ase format to the same directory as the log file was loaded from. That model can be directly imported into the UnrealEditor Generic Browser, then if placed into the map, at location 0,0,0 it will direct illustrate the player path through the level. This will also be visible if the map is played.

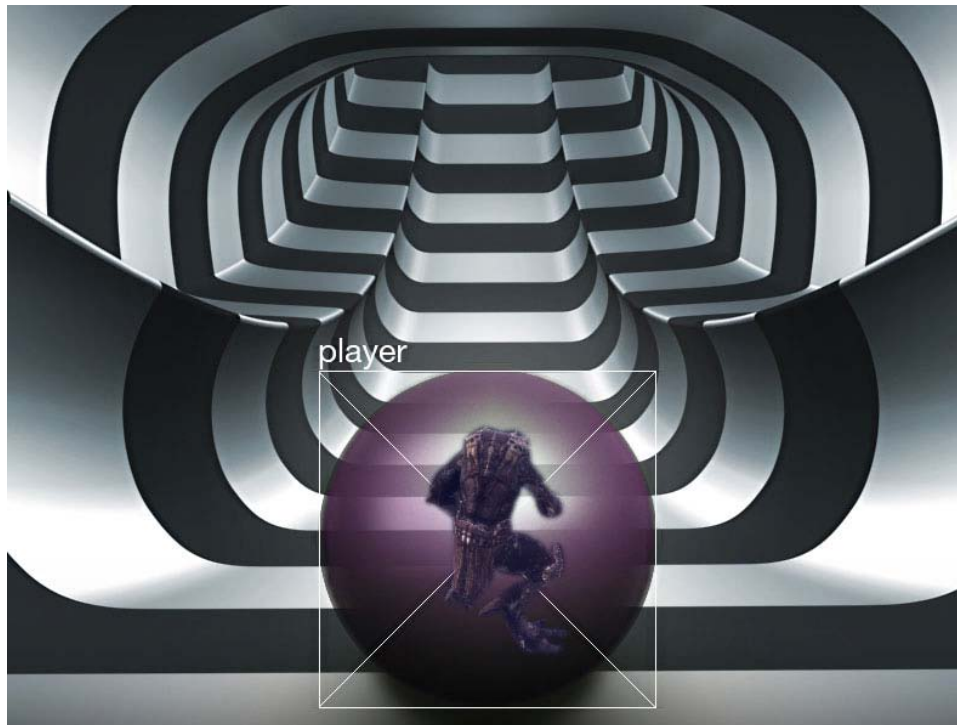
Tech Note:

When importing model at setting to 0,0,0 it is sometimes required to mirror the Y-Axis to get the path to line up correctly with the map. *Bug has been Logged*

Camera

Overview

Curvature uses a 3rd person camera. While in Bella form the camera is hard attached to the head of your character Bella so when she turns her head the view will always follow where she is looking. The camera is positioned four feet behind Bella and 3 feet above her. The camera will move closer to Bella if the camera's position collides with world geometry or meshes. While in either of the fields, the camera is a follow cam allowing it to track the player's movement in both the horizontal and vertical planes. However the vertical plane is limited in both rate of change and total range of motion preventing the follow cam from ever raising or lowering the player's view past 45° from flat.



Camera Features

- 3rd person
- Follow, horizontal and vertical
- Reset button (move to direction of player travel)
- Follow disables during manual input from player

Cinematic/Cutscene

During all cut sequences and cinematic views, the Interface/HUD will be removed.

Technical Implementation Plan

We will inherit the default 3D Behindview camera from the Unreal engine as a starting point.

The Game World

Overview

The entire gameworld of Curvature is rounded. Textures are to be simple with most of the heavy lifting done via a custom shader that we will be creating. For more detailed information the general look and feel of the world, see the ART BIBLE.



Technical Implementation Plan

The world will be created with models produced in 3DS Max and imported into the Unreal Editing tool

Movement:

Bella and the Shades movement will be key frame animated and then imported into the Unreal Engine's animation tree.

Time Estimation is Rigging 8 hours, and 60 hours Animating for Bella; and 8 hours Rigging, with 50 hours of Animation tree work for the Shades

Game Characters

Overview

Curvature only has one character required for the actual gameplay, Bella. Any other characters would be imported for use in cinematics or rendering them off for promo art, however no plans have been made to rig and animate any others besides Bella.

Shader

A custom shader will be used for all the characters, it will allow for details to be pushed with little input.

Protagonist

- Name: Bella
- Appearance: A young girl, aprox. age 15.
- Title: The Marked one (the Bearer of the Device)
- Attributes: Mortal, natural
- Weapon: The Device

Technical Implementation

Shader

All the characters in the game will use our character shader model, this model includes taking an AO and a normal map and combining them in a simple custom lighting model to display the details of our high res base geo on the low res in-game assets for the game.

Protagonist

The protagonist, Bella will require a custom rig and animations to go with it. These animations will include jump, walk, run, land, mid air idle, an “in the field” idle animation.

Antagonist

Lodi requires nothing to be made asset wise for gameplay, a simply mesh for use in cinematics could be used. Rigging and animation not required for character.

Allies

Magnus requires nothing to be made asset wise for gameplay, a simply mesh for use in cinematics could be used. Rigging and animation not required for character.

Engine

Letter 6 will be utilizing the Unreal Engine 3 from Epic.

Unreal Editor 3

Folder structure - SEE NAMING CONVENTIONS

All 3D Assets are to be created in power of two for unit size and on grid

(Epic suggests packages less than 100MB (From Lahn))

Technical Requirements

Animation Pipeline for Unreal Editor for Unreal Engine 3

In-game Cinematic workflow with Matinee

Naming Conventions Script

QA Testing import script workflow for Unreal ED

Technical Risks

Custom Animations

Because we are using custom meshes which are of odd proportions we are forced to create custom rigs and animate them for all the character meshes in the game. Unreal Editor animation pipeline information required.

Physics Solving

All of the bubble interaction in our game relies heavily on the solving that the physics engine built into Unreal Tournament 3 has. (Havok) This has the potential to be problematic for testing, let alone distribution, consistent behaviour of the bubble is what is desired, however with all the solving happening real-time and in a unique fashion every time. This appears to manifest itself most severely by having the fields "punch through" solid geometry.

Additional Specs

Folder Structure

Sound Library

Characters

Bella
Magnus
Device
Misc

Buildings

Misc
Ambient

3d Assets

Characters

Bella

Textures

Magnus

Textures

Device

Textures

Misc

Textures

Buildings

Textures

Level Design

Textures

Development Fork (Daily Wip)

Published

- Cooked PC
 - Custom Maps
 - Founders Building
 - City
 - Power Station

SRC

- Core
- Editor
- LETTER6
 - Classes

Unpublished

- CookedPC
 - Custom Maps
 - Letter6
 - Characters
 - Effects
 - Environments
 - Maps
 - Music
 - Pickups
 - Sounds
 - UI
 - Weapons

Daily Build Fork (Current Production)

Yesterday Build Fork (Previous Daily)

Naming Conventions

Meshes

example(s):

t6_MSH_CHR_Bella_v001_EF

t6_MSH_ENV_PalmTree03_v001_SP

t6_MSH_N%_V%_T%

- N% - Name of Mesh
- CHR = Character
- ENV = Environment
- WPN = Weapon
- V% - Version Number (Version 1 = v001, Version 2 = v002)
- T% - Tag (Evan Fiddes = EF, Scott Prince = SP)

Textures

example(s):

t6_TEX_CHR_Bella_DIF_v001_EF

t6_TEX_ENV_PalmTree03_NOR_v001_SP

t6_TEX_ENV_PalmTree03_ALP_v005_SP

t6_TEX_N%_M%_V%_T%

- N% - Name of Texture
- CHR = Character
- ENV = Environment
- WPN = Weapon
- M% - Map Type (Diffuse = DIF, Normal = NOR, Alpha = ALP)
- V% - Version Number (Version 1 = v001, Version 2 = v002)
- T% - Tag (Evan Fiddes = EF, Scott Prince = SP)

Levels

example(s):

T6_LVL_01_FoundersBuilding_v002_EF

t6_LVL_L%_N%_V%_T%

- L% - Level Index Number (First Level = 01, Second Level = 02)
- N% - Name of Level (Saloon, Jailhouse)
- V% - Version Number (Version 1 = v001, Version 2 = v002)
- T% - Tag (Evan Fiddes = EF, Scott Prince = SP)

Audio

example(s):

T6_AUD_SFX_BellaScream2_v002_SP

T6_AUD_MUS_LodiBossEnd2_v006_SP

t6_AUD_C%_N%_V%_T%

- C% - Audio Class
- SFX - SoundFX
- ABL - Ambient Loop
- MUS - Music
- N% - Name of Audio File
- V% - Version Number (Version 1 = v001, Version 2 = v002)
- T% - Tag (Evan Fiddes = EF, Scott Prince = SP)

Testing

Overview

Team Letter 6's goals for QA are to have the following implemented for production through to final:

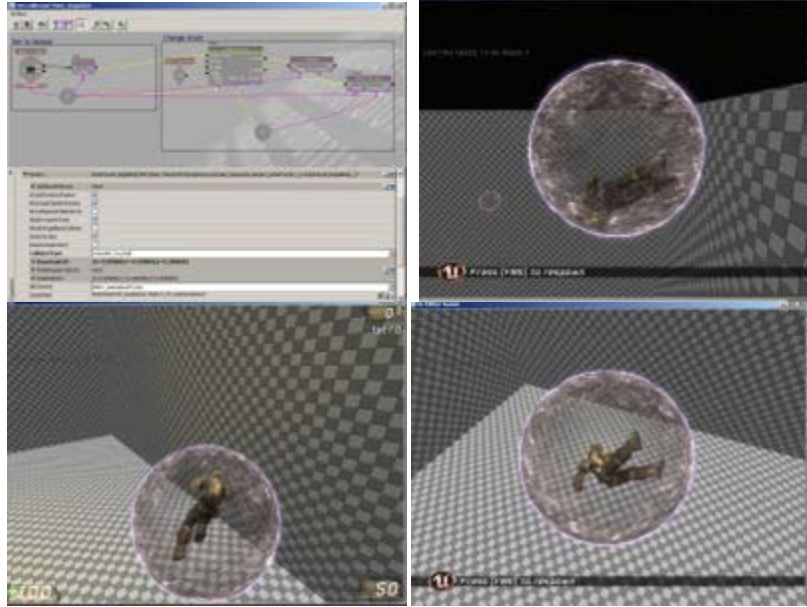
- Bug Tracking - a web based tool to track and communicate issues with the game
- Open testing - a machine that can be used at a moment's notice that will always have the current production build
- Change Control - a system in place to control and implement change
- Focus testing - a method to ensure the high level game play goals are being met ie. level progression, flow, visual cues

See our detailed QA Test plan for more information.

POC

Early Prototype

Screen shots from our earliest build.



What we did:

- Spawn an Interpret Actor (Ridged Body Mesh) and -Attach- it to the players location.
- Set world Gravity to .01 (Low Gravity) in the World Properties tab, under View in menu.
- Set a global gravity volume to ADD gravity back into the map. (So the map Gravity seems normal)
- Set a trigger to destroy the Interpret Actor on command. (toggle -Bubble- on or off)

What we planned to do:

- Make custom Actor class (Child of UTPawn) that isn't effected by My-Custom-Gravity-Volume. (So it floats!)
- Create a custom Volume that doesn't effect my New-UTPawn-Actor-Type. (So I can control Who or What Floats)
- Create a trigger that switches Actors to my New-UTPawn-Actor-Type so that they will float in Low Gravity.

Limitations / Success:

The entire prototype was made in kismet. None of the functionality was useable in the final build. Only "Light Field" mechanic featured in the demo. We successfully accomplished the goal of proving floating bubble mechanic.