Design Document for:

Escape from Tibet

Written by:
Bilal Ghalib
Aaron Curley

C# Game Programming
CIS 297
Dr. Bruce Elenbogen

Version: 003

April 17, 2008

Table of Contents

Revision History

Version	Date	Revision Author	Details
001	3/13/08	Aaron and Bilal	First draft for game plan.
002	4/11/08	Aaron Curley	Expansion of the document to complete it to match the current game design and to add all the stuff needed for final submission. 3 UML diagrams added. Massive restructuring. UI section added. Game play section redone.
003	4/17/08	Aaron Curley	Update to UML class diagram.

1. Game Overview

1.1. Story Abstract

James Bond has just recovered an important government document detailing the time and place of an attack on a U. S. embassy from the evil Dr. Soothsayer's lab high atop a mysterious unnamed mountain in Tibet. To escape this treacherous place, he must ski down the steep slopes, populated with obstacles such as rocks, trees, and snow drifts. He needs to reach his car (at the base of the mountain) before time runs out and the explosion proves fatal to the embassy workers.

1.2. Genre

Escape from Tibet is an arcade-style game suitable for any person who has coordinated use of a mouse.

1.3. Target Audience

Since the graphics in the game have an animated look to them, *Escape from Tibet* targets those of all ages. Furthermore, the game's alternative user input allows users who are unable to use the mouse or keyboard to still enjoy playing the game by using head movements to control Mr. Bond.

1.4. Game Flow Summary

Escape from Tibet is played as most would expect from a 2D game. The game first presents the main menu to the player. The player can use the mouse, keyboard, and possibly head motions to navigate through the menu system. When the user chooses the "start game" option, a list of choices for user input are presented, allowing the user to choose which input device to use while playing the game. The valid choices are:

- Camera Input
- Mouse Input
- Keyboard Input

Once a selection is made, the game begins and the user sees the slope populated with obstacles.

While playing the game, the user controls the player as he skies down the slope. Once the game ends, the player can return to the main menu by clicking or pressing spacebar/enter. The user can also prematurely return to the main menu by pressing ESC at any time.

1.5. Appearance

Escape from Tibet is a 3rd person game using a 2D graphics engine. The "camera" remains fixed behind Mr. Bond at most times, simulating a 3D view of the slope below and any obstacles as seen by Mr. Bond. However, if Mr. Bond trips, the "camera" seems to "detach" from behind Mr. Bond's head, and the player sees Mr. Bond tripping and falling to the ground, after which the "camera" is returned its position behind Mr. Bond's head.

1.5.1. Screen Mockup of the Perspective View

2. Game Specification

2.1. Game Play

The user controls Mr. Bond's movements as he skis down the hill. The player must ensure that Mr. Bond arrives safely but quickly, to reach his car from which he will transmit a warning to U.S. security personnel about the pending blast.

Control of Mr. Bond in *Escape from Tibet* can be achieved through the use of a mouse, keyboard, or through the use of head movements recorded using a webcam attached to the player's computer.

Player Actions:

Action	Mouse Movement	Head Movement	Keyboard
Move left	**Mouse moved to the left.**	**Player leans to the left.**	**Left arrow key.**
Move right	**Mouse moved to the right.**	**Player leans to the right.**	**Right arrow key.**
Increase speed	**Mouse moved upward.**	**Player leans forward.**	**Up arrow key.**
Decrease speed	**Mouse moved downward.**	**Player leans back.**	**Down arrow key.**
Jump	**Left mouse click.**	**Player moves his head quickly forward. (Player can also use the space bar.)**	**Space bar.**

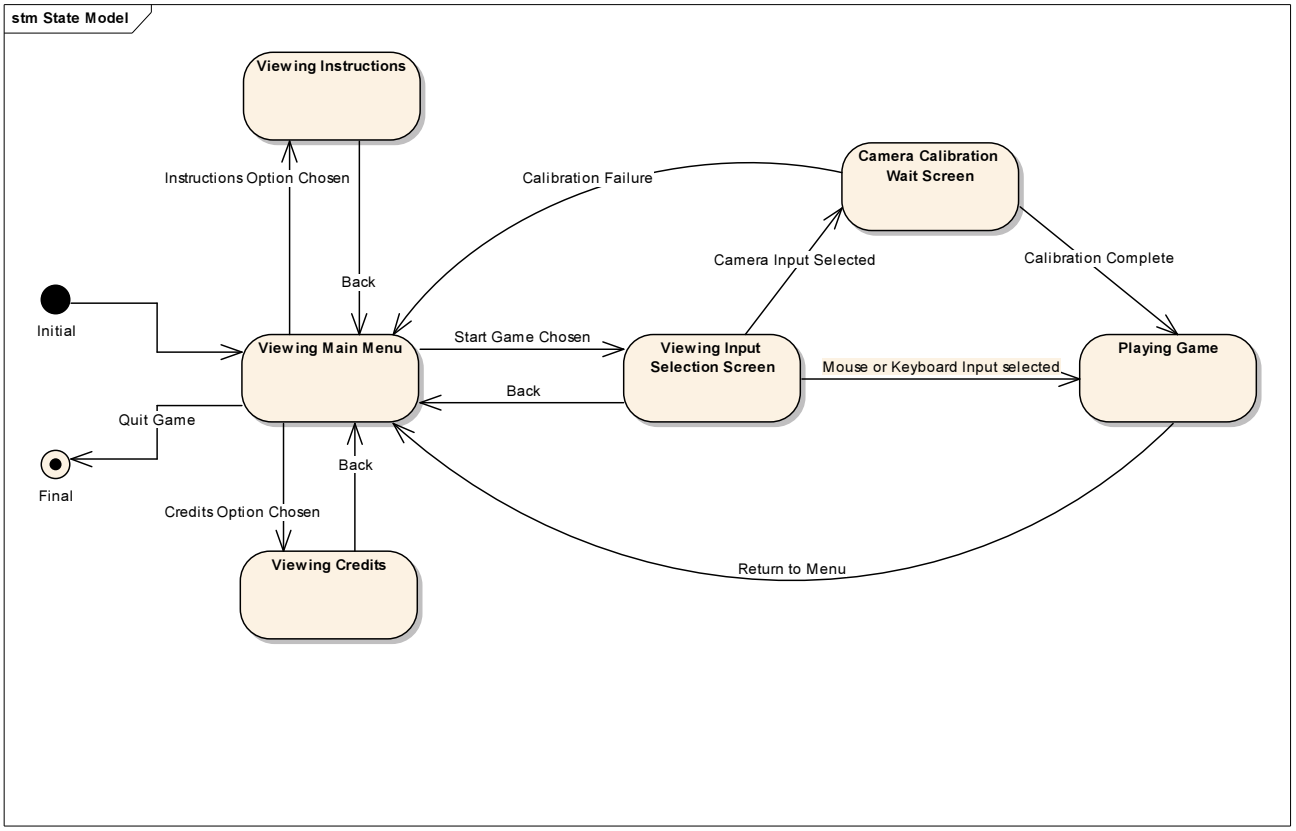
There are two types of obstacles. Those that Mr. Bond trips over and those that act as "bumps" where Mr. Bond travels over them, as if they were a "bump" on the course. Mr. Bond takes no damage by traveling over a "bump" object in the course. In fact, such "bump" objects can be used to make Mr. Bond jump even higher, by timing his jump at the peak of the "bump". However, traveling over a bump entails some degree of risk; Mr. Bond remains in the air for a short time after hitting a bump at high speeds. Even a few moments without control of his current direction can result in a nasty collision between Mr. Bond and a nearby tree.

Mr. Bond loses health when he hits an object that makes him trip. The amount of damage taken depends on the energy of the collision; faster speeds result in higher damages. Mr. Bond starts out with 100 health. Once his health reaches zero, he dies, and does not complete his mission.

Jumping can be used to avoid shorter obstacles. However, such an action is risky because if the jump is timed incorrectly, Mr. Bond's skis can become entangled in the obstacle and cause him to trip.

2.2. Interface Description

2.2.1. Screen Flow State Diagram



2.2.2. Menu Interface

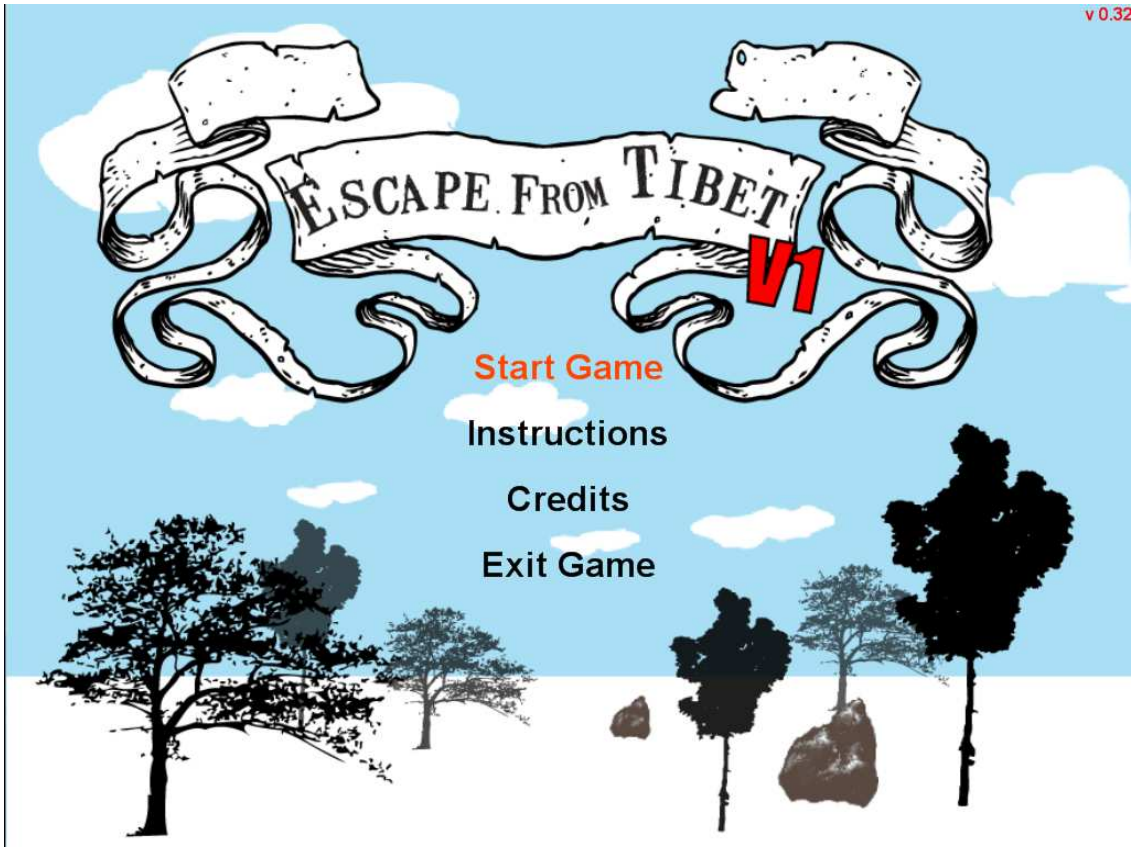
The menu interface presents a title screen to the user and allows the user to start the game with the desired options or to get information on how to play. There are a number of screens included in the menu interface, and they are described here.

For all menus, the current "choice" selected by the user is highlighted in a yellow color. This allows the user to identify the choice that will be selected if ENTER is pressed (to allow for keyboard navigation). The user may also click on a choice using the mouse.

2.2.2.1. Main Menu

The main menu allows the user to navigate to the other available screens. Possible choices are:

- **Start Game** – Begins the process of starting a game by displaying the input selection screen.
- **Instructions** – Displays the instructions screen.
- **Credits** – Displays the credits screen.
- **Exit Game** – Exists *Escape from Tibet*.



2.2.2.2. Instructions Screen

This screen simply displays text to the user that instructs him how to play the game. There is a single option on this screen labeled "Back to Main Menu." This returns the user to the main menu.

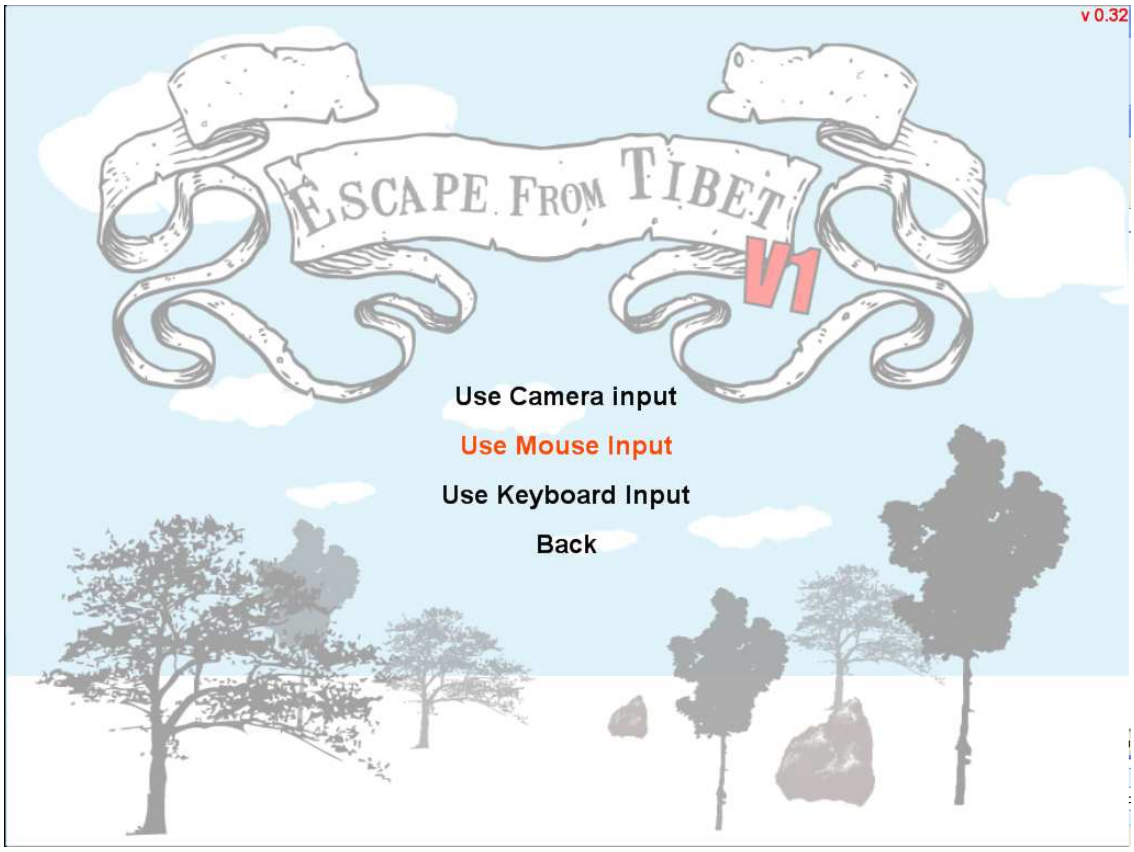
2.2.2.3. Credits Screen

This screen simply displays the credits text to the user. There is a single option on this screen labeled "Back to Main Menu." This returns the user to the main menu.

2.2.2.4. Input Selection Screen

This screen allows the user to choose which input device to use while playing the game. The choices on this menu are:

- **Use Camera Input** – Begins the process of initializing and calibrating the camera.
- **Use Mouse Input** – Starts the game, using mouse input to control Mr. Bond.
- **Use Keyboard Input** – Starts the game, using keyboard input to control Mr. Bond.
- **Back** – Returns to the main menu.



2.2.2.5. Camera Calibration Screen

The camera calibration screen is displayed to the user if the "Use Camera Input" option is selected. The game remains on this screen until the camera is successfully initialized, at which point, the user is transferred to the game. If the camera fails to initialize, an error message is displayed on the screen. Once the user clicks the mouse or presses SPACE BAR / ENTER, the user is returned to the main menu.

2.2.3. In-Game HUD

The in-game HUD is simplistic and uses text to present information to the user.

Information listed includes:

- **Percentage of slope traveled** – How much of the course has been completed.
- **Time** – How much time has elapsed.
- **Health** – Mr. Bond's current health.

The user may at any time press the ESC button to quit and return to the main menu. If the player successfully completes the course or takes too much damage and dies, the game switches back to the main menu when the user presses ESC, SPACE, ENTER, or clicks the left mouse button.

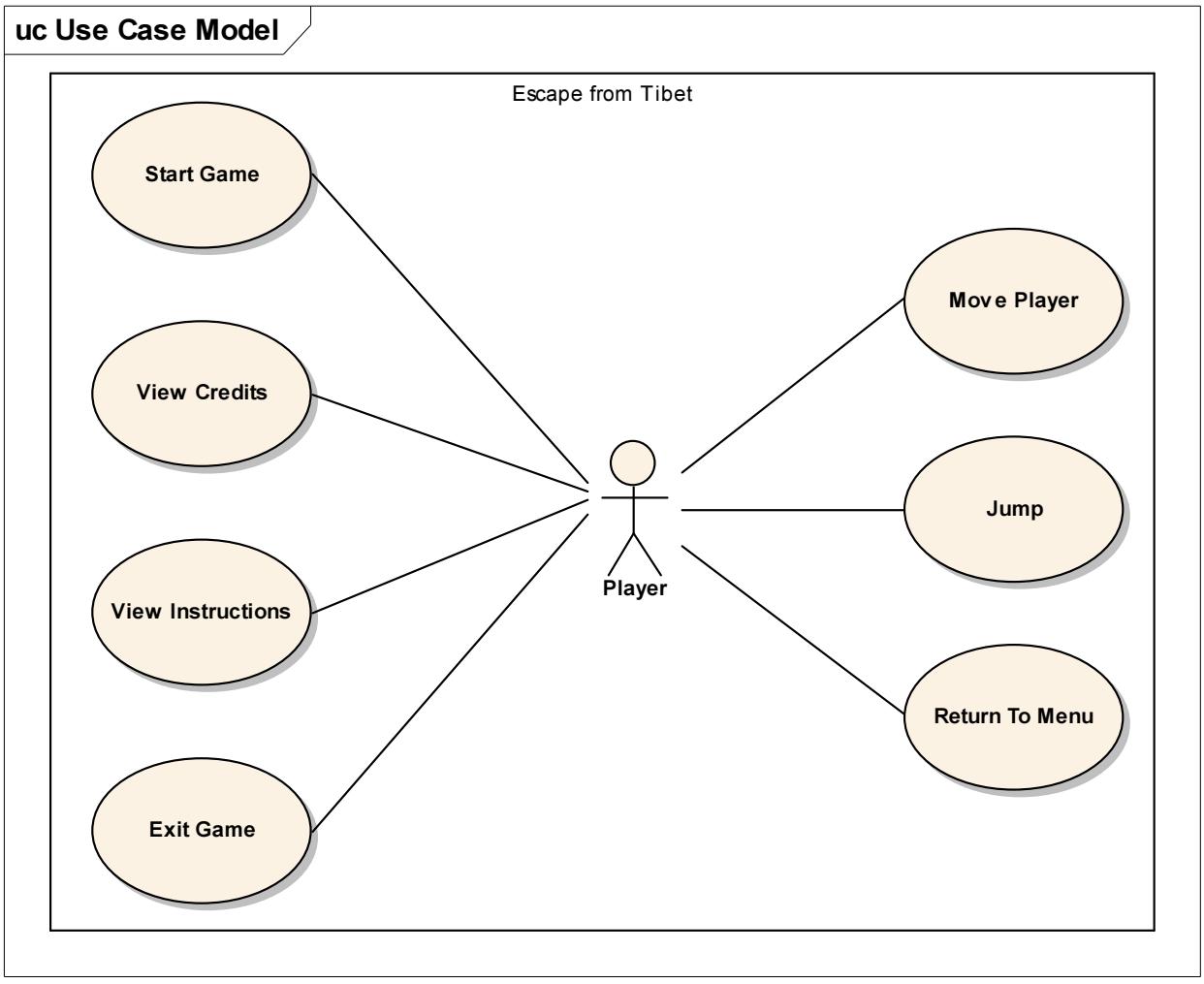
Screen Shot of the HUD:

2.3. Course Generation

The length and width of the in-game course are defined as constants in the program. The course "wraps" along the horizontal axis; therefore, moving the player far to the left or far to the right causes the course to "wrap around." This way, the player can never "run out" of course by moving too far to the left or right. Once the player exceeds the length of the course (has traveled the entire course distance), the player wins and sees a congratulatory message.

Course objects are generated and placed randomly on the course when *Escape from Tibet* is loaded. Once the game is exited and restarted, objects will be in different locations than before. The number of objects to generate is also programmed in as a constant.

2.4. Use Case Diagram



3. Technical Information

3.1. Development Specification

Escape from Tibet will be developed in Visual Studio 2005 using XNA Game Studio 2.0. The system requirements for playing *Escape from Tibet* are as follows:

- Pentium III or AMD Athlon 1.0 GHz processor
- 256MB RAM
- DirectX® 9.0c compatible graphics card with shader 1.1 support or greater.
- Microsoft .NET Framework version 2.0.
- Microsoft XNA Framework 2.0 Redistributable.

3.2. Project Team Description / Timeline

3.2.1. Team Name

"Team Vision"

3.2.2. Production Team Members

Aaron Curley – Game design, game engine design/implementation, graphics.
Bilal Ghalib – Game design, camera interface design/implementation, sound, graphics.

Special thanks to Ruth Curley for assistance with graphics.

3.2.3. Activity Timeline

First week:

- OpenCV proxy interacting with C# game engine code.
- Game engine architecture designed.
- Preliminary menu code.
- Mouse input and camera input utilized on menu.

Second week:

- Head position HUD implemented.
- Head position calibration screen working.
- Game engine rendering landscape.
- Left-right movement implemented.
- Working game timer.

Third week:

- Obstacle objects positioned and moving properly to simulate forward movement.
- Initial obstacle collisions.
    - Fatal.
    - Nonfatal.
- Initial sound effects.

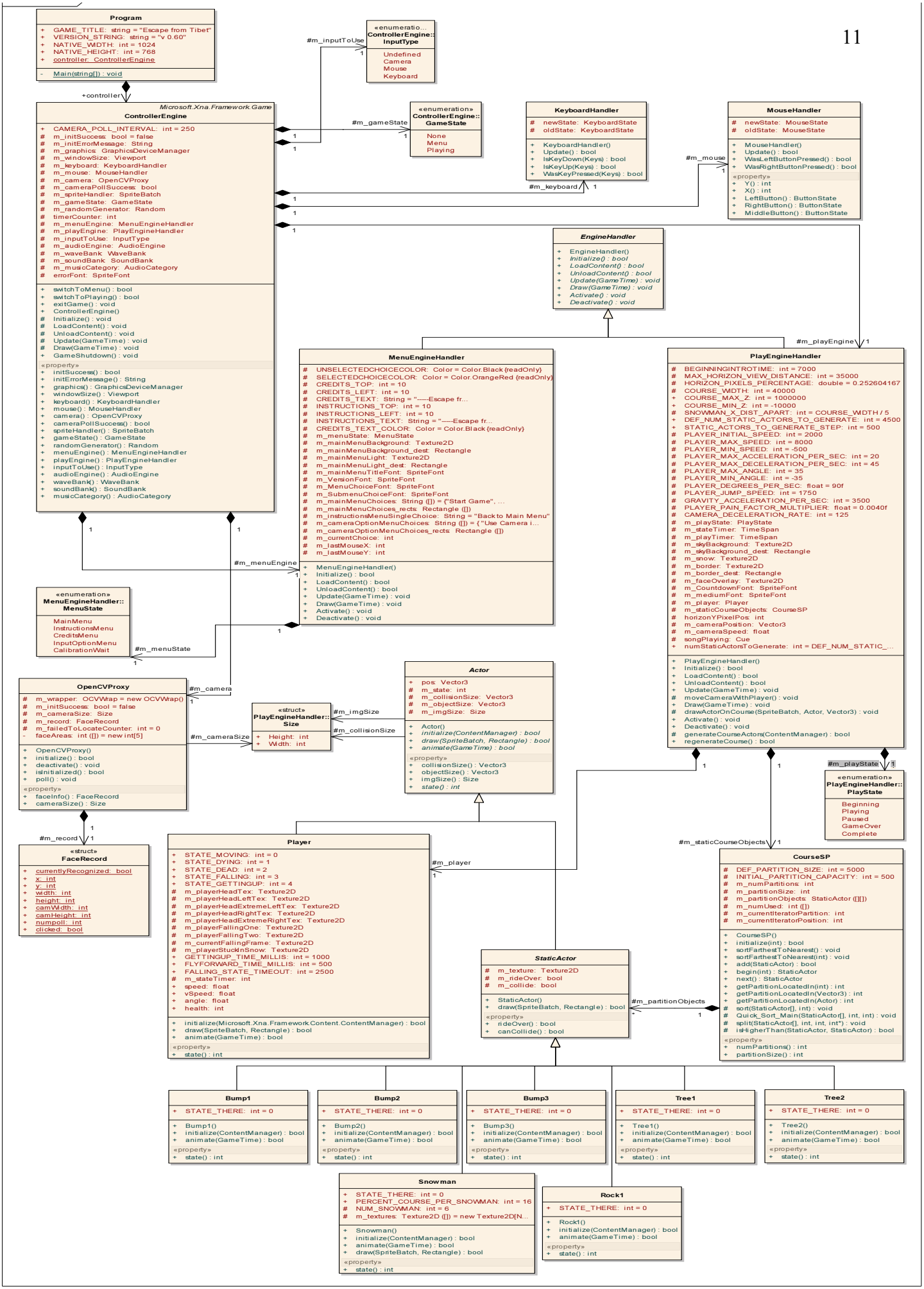
Fourth week:

- Finish collision tweaking.

- Add further sound effects.
- Testing.
- Packaging for final release.

3.3. Class Diagram

A UML class diagram created using Enterprise Architect is included on the next page. This diagram illustrates the architecture of *Escape from Tibet*.


Program
+ GAME_TITLE: string = "Escape from Tibet"
+ VERSION_STRING: string = "v 0.60"
+ NATIVE_WIDTH: int = 1024
+ NATIVE_HEIGHT: int = 768
+ controller: ControllerEngine
- Main(string[]) : void
+controller
«enumeratio... ControllerEngine:: InputType
Undefined
Camera
Mouse
Keyboard
#m_inputToUse
Microsoft.Xna.Framework.Game
ControllerEngine
+ CAMERA_POLL_INTERVAL: int = 250
# m_initSuccess: bool = false
# m_initErrorMessage: String
# m_graphics: GraphicsDeviceManager
# m_windowSize: Viewport
# m_keyboard: KeyboardHandler
# m_mouse: MouseHandler
# m_camera: OpenCVProxy
# m_cameraPollSuccess: bool
# m_spriteHandler: SpriteBatch
# m_gameState: GameState
# m_randomGenerator: Random
# timerCounter: int
# m_menuEngine: MenuEngineHandler
# m_playEngine: PlayEngineHandler
# m_inputToUse: InputType
# m_audioEngine: AudioEngine
# m_waveBank: WaveBank
# m_soundBank: SoundBank
# m_musicCategory: AudioCategory
# errorFont: SpriteFont
+ switchToMenu() : bool
+ switchToPlaying() : bool
+ exitGame() : void
+ ControllerEngine()
# Initialize() : void
# LoadContent() : void
# UnloadContent() : void
# Update(GameTime) : void
# Draw(GameTime) : void
+ GameShutdown() : void
«property»
+ initSuccess() : bool
+ initErrorMessage() : String
+ graphics() : GraphicsDeviceManager
+ windowSize() : Viewport
+ keyboard() : KeyboardHandler
+ mouse() : MouseHandler
+ camera() : OpenCVProxy
+ cameraPollSuccess() : bool
+ spriteHandler() : SpriteBatch
+ gameState() : GameState
+ randomGenerator() : Random
+ menuEngine() : MenuEngineHandler
+ playEngine() : PlayEngineHandler
+ inputToUse() : InputType
+ audioEngine() : AudioEngine
+ waveBank() : WaveBank
+ soundBank() : SoundBank
+ musicCategory() : AudioCategory
#m_gameState
«enumeration» ControllerEngine:: GameState
None
Menu
Playing
KeyboardHandler
newState: KeyboardState
oldState: KeyboardState
+ KeyboardHandler()
+ Update() : bool
+ IsKeyDown(Keys) : bool
+ IsKeyUp(Keys) : bool
+ WasKeyPressed(Keys) : bool
#m_keyboard
MouseHandler
newState: MouseState
oldState: MouseState
+ MouseHandler()
+ Update() : bool
+ WasLeftButtonPressed() : bool
+ WasRightButtonPressed() : bool
«property»
+ Y() : int
+ X() : int
+ LeftButton() : ButtonState
+ RightButton() : ButtonState
+ MiddleButton() : ButtonState
#m_mouse
EngineHandler
+ EngineHandler()
+ Initialize() : bool
+ LoadContent() : bool
+ UnloadContent() : bool
+ Update(GameTime) : void
+ Draw(GameTime) : void
+ Activate() : void
+ Deactivate() : void
#m_playEngine
MenuEngineHandler
UNSELECTEDCHOICECOLOR: Color = Color.Black {readOnly}
SELECTEDCHOICECOLOR: Color = Color.OrangeRed {readOnly}
CREDITS_TOP: int = 10
CREDITS_LEFT: int = 10
CREDITS_TEXT: String = "-----Escape fr...
INSTRUCTIONS_TOP: int = 10
INSTRUCTIONS_LEFT: int = 10
INSTRUCTIONS_TEXT: String = "-----Escape fr...
CREDITS_TEXT_COLOR: Color = Color.Black {readOnly}
m_menuState: MenuState
m_mainMenuBackground: Texture2D
m_mainMenuBackground_dest: Rectangle
m_mainMenuLight: Texture2D
m_mainMenuLight_dest: Rectangle
m_mainMenuTitleFont: SpriteFont
m_VersionFont: SpriteFont
m_MenuChoiceFont: SpriteFont
m_SubmenuChoiceFont: SpriteFont
m_mainMenuChoices: String ([]) = {"Start Game", ...
m_mainMenuChoices_rects: Rectangle ([])
m_instructionsMenuSingleChoice: String = "Back to Main Menu"
m_cameraOptionMenuChoices: String ([]) = { "Use Camera i...
m_cameraOptionMenuChoices_rects: Rectangle ([])
m_currentChoice: int
m_lastMouseX: int
m_lastMouseY: int
+ MenuEngineHandler()
+ Initialize() : bool
+ LoadContent() : bool
+ UnloadContent() : bool
+ Update(GameTime) : void
+ Draw(GameTime) : void
+ Activate() : void
+ Deactivate() : void
#m_menuEngine
PlayEngineHandler
BEGINNINGINTROTIME: int = 7000
MAX_HORIZON_VIEW_DISTANCE: int = 35000
HORIZON_PIXELS_PERCENTAGE: double = 0.252604167
COURSE_WIDTH: int = 40000
+ COURSE_MAX_Z: int = 1000000
+ COURSE_MIN_Z: int = -10000
# SNOWMAN_X_DIST_APART: int = COURSE_WIDTH / 5
+ DEF_NUM_STATIC_ACTORS_TO_GENERATE: int = 4500
+ STATIC_ACTORS_TO_GENERATE_STEP: int = 500
# PLAYER_INITIAL_SPEED: int = 2000
# PLAYER_MAX_SPEED: int = 8000
# PLAYER_MIN_SPEED: int = -500
# PLAYER_MAX_ACCELERATION_PER_SEC: int = 20
# PLAYER_MAX_DECELERATION_PER_SEC: int = 45
# PLAYER_MAX_ANGLE: int = 35
# PLAYER_MIN_ANGLE: int = -35
# PLAYER_DEGREES_PER_SEC: float = 90f
# PLAYER_JUMP_SPEED: int = 1750
# GRAVITY_ACCELERATION_PER_SEC: int = 3500
# PLAYER_PAIN_FACTOR_MULTIPLIER: float = 0.0040f
# CAMERA_DECELERATION_RATE: int = 125
# m_playState: PlayState
# m_stateTimer: TimeSpan
# m_playTimer: TimeSpan
# m_skyBackground: Texture2D
# m_skyBackground_dest: Rectangle
# m_snow: Texture2D
# m_border: Texture2D
# m_border_dest: Rectangle
# m_faceOverlay: Texture2D
# m_CountdownFont: SpriteFont
# m_mediumFont: SpriteFont
# m_player: Player
# m_staticCourseObjects: CourseSP
# horizonYPixelPos: int
# m_cameraPosition: Vector3
# m_cameraSpeed: float
# songPlaying: Cue
+ numStaticActorsToGenerate: int = DEF_NUM_STATIC_...
+ PlayEngineHandler()
+ Initialize() : bool
+ LoadContent() : bool
+ UnloadContent() : bool
+ Update(GameTime) : void
# moveCameraWithPlayer() : void
+ Draw(GameTime) : void
# drawActorOnCourse(SpriteBatch, Actor, Vector3) : void
+ Activate() : void
+ Deactivate() : void
# generateCourseActors(ContentManager) : bool
+ regenerateCourse() : bool
«enumeration» MenuEngineHandler:: MenuState
MainMenu
InstructionsMenu
CreditsMenu
InputOptionMenu
CalibrationWait
#m_menuState
OpenCVProxy
m_wrapper: OCVWrap = new OCVWrap()
m_initSuccess: bool = false
m_cameraSize: Size
m_record: FaceRecord
m_failedToLocateCounter: int = 0
- faceAreas: int ([]) = new int[5]
+ OpenCVProxy()
+ initialize() : bool
+ deactivate() : void
+ isInitialized() : bool
+ poll() : void
«property»
+ faceInfo() : FaceRecord
+ cameraSize() : Size
#m_camera
#m_cameraSize
«struct» PlayEngineHandler:: Size
+ Height: int
+ Width: int
#m_imgSize
#m_collisionSize
Actor
+ pos: Vector3
# m_state: int
# m_collisionSize: Vector3
# m_objectSize: Vector3
# m_imgSize: Size
+ Actor()
+ initialize(ContentManager) : bool
+ draw(SpriteBatch, Rectangle) : bool
+ animate(GameTime) : bool
«property»
+ collisionSize() : Vector3
+ objectSize() : Vector3
+ imgSize() : Size
+ state() : int
#m_playState
«enumeration» PlayEngineHandler:: PlayState
Beginning
Playing
Paused
GameOver
Complete
«struct» FaceRecord
+ currentlyRecognized: bool
+ x: int
+ y: int
+ width: int
+ height: int
+ camWidth: int
+ camHeight: int
+ numpoll: int
+ clicked: bool
#m_record
Player
+ STATE_MOVING: int = 0
+ STATE_DYING: int = 1
+ STATE_DEAD: int = 2
+ STATE_FALLING: int = 3
+ STATE_GETTINGUP: int = 4
# m_playerHeadTex: Texture2D
# m_playerHeadLeftTex: Texture2D
# m_playerHeadExtremeLeftTex: Texture2D
# m_playerHeadRightTex: Texture2D
# m_playerHeadExtremeRightTex: Texture2D
# m_playerFallingOne: Texture2D
# m_playerFallingTwo: Texture2D
# m_currentFallingFrame: Texture2D
# m_playerStuckInSnow: Texture2D
+ GETTINGUP_TIME_MILLIS: int = 1000
+ FLYFORWARD_TIME_MILLIS: int = 500
+ FALLING_STATE_TIMEOUT: int = 2500
# m_stateTimer: int
+ speed: float
+ vSpeed: float
+ angle: float
+ health: int
+ initialize(Microsoft.Xna.Framework.Content.ContentManager) : bool
+ draw(SpriteBatch, Rectangle) : bool
+ animate(GameTime) : bool
«property»
+ state() : int
#m_player
StaticActor
m_texture: Texture2D
m_rideOver: bool
m_collide: bool
+ StaticActor()
+ draw(SpriteBatch, Rectangle) : bool
«property»
+ rideOver() : bool
+ canCollide() : bool
#m_partitionObjects
#m_staticCourseObjects
CourseSP
DEF_PARTITION_SIZE: int = 5000
INITIAL_PARTITION_CAPACITY: int = 500
m_numPartitions: int
m_partitionSize: int
m_partitionObjects: StaticActor ([][])
m_numUsed: int ([])
m_currentIteratorPartition: int
m_currentIteratorPosition: int
+ CourseSP()
+ initialize(int) : bool
+ sortFarthestToNearest() : void
+ sortFarthestToNearest(int) : void
+ add(StaticActor) : bool
+ begin(int) : StaticActor
+ next() : StaticActor
+ getPartitionLocatedIn(int) : int
+ getPartitionLocatedIn(Vector3) : int
+ getPartitionLocatedIn(Actor) : int
# sort(StaticActor[], int) : void
# Quick_Sort_Main(StaticActor[], int, int) : void
# split(StaticActor[], int, int, int*) : void
# isHigherThan(StaticActor, StaticActor) : bool
«property»
+ numPartitions() : int
+ partitionSize() : int
Bump1
+ STATE_THERE: int = 0
+ Bump1()
+ initialize(ContentManager) : bool
+ animate(GameTime) : bool
«property»
+ state() : int
Bump2
+ STATE_THERE: int = 0
+ Bump2()
+ initialize(ContentManager) : bool
+ animate(GameTime) : bool
«property»
+ state() : int
Bump3
+ STATE_THERE: int = 0
+ Bump3()
+ initialize(ContentManager) : bool
+ animate(GameTime) : bool
«property»
+ state() : int
Tree1
+ STATE_THERE: int = 0
+ Tree1()
+ initialize(ContentManager) : bool
+ animate(GameTime) : bool
«property»
+ state() : int
Tree2
+ STATE_THERE: int = 0
+ Tree2()
+ initialize(ContentManager) : bool
+ animate(GameTime) : bool
«property»
+ state() : int
Snowman
+ STATE_THERE: int = 0
+ PERCENT_COURSE_PER_SNOWMAN: int = 16
# NUM_SNOWMAN: int = 6
# m_textures: Texture2D ([]) = new Texture2D[N...
+ Snowman()
+ initialize(ContentManager) : bool
+ animate(GameTime) : bool
+ draw(SpriteBatch, Rectangle) : bool
«property»
+ state() : int
Rock1
+ STATE_THERE: int = 0
+ Rock1()
+ initialize(ContentManager) : bool
+ animate(GameTime) : bool
«property»
+ state() : int