Vacare

A Simple Puzzle Game Created

Using Microsoft XNA

Chris Ragland

CS 470 Write-up

12/15/2008

**Table of Contents**

Abstract 3

1. Introduction 3
2. Project Overview 3
	1. Game Data 4
	2. Level Data 4
3. Project Requirements 5
	1. General Requirement Definitions 5
	2. In Game Requirement Definitions 6
	3. Optional Requirement Definitions 6
	4. System Specifications 6
4. System Design 6
	1. Data Structures 8
	2. Algorithms 9
5. Development Process 9
	1. Work Breakdown 10
6. Results 12
7. Conclusion 12

Appendix A: User Manual - Attached

Appendix B: Project Files/Source Code - Attached

**A Puzzle Game Created in XNA**

**Chris Ragland**

**Abstract**

Microsoft XNA is a set of tools provided to make game creation simple and accessible for everyone. Games created with the XNA Framework are able to run on multiple platforms, PC, Xbox 360, and Microsoft Zune. Games created for the Xbox 360 in the XNA Framework can also be published and distributed through Xbox Live, making them available for purchase. This project uses the XNA Framework to create a PC game. The game is a simple puzzle game that contains 15 levels, tracks high scores, plays sound effects, and other things a user would expect from an arcade puzzle game.

1. **Introduction**

When deciding what to do for my Senior Project I stumbled across the XNA tools. I looked at the tools for a few weeks and tried to work with them and liked what I saw. The potential to be able to not only port my final creation over to an Xbox 360, but also to be able to sell anything that I was able to create to people who would want to use it was also another driving force in my selection of creating a game using the XNA Framework for my project.

Because I was the end customer for this project, I was able to alter the requirements at some points in the project where a normal customer might not have allowed changes. These changes mostly occurred due to quirks with the XNA set of tools that I didn't know before getting involved with them, including the shading language supported, the rendering of rotating images, and other things.

While the project may not fall exactly in line with the requirements that I set forward in my proposal, the end product is very close to what I was wanting, and is in a finished state. The project should show what can be possible given the XNA tools, a limited workforce, and a short period of time.

1. **Project Overview**

The game I ended up creating is a twist on simple puzzle shooter games. In most puzzle shooters you are tasked with destroying a set number of targets to progress to the next level. The same holds for my project, with the twist being that the playing area can experience different gravity forces and that alters the path the bullet is traveling along. The game also has other features like a title screen and high score tracking to give the player a feeling that they really are playing a game.

* 1. **Game Data**

The game being created originally for an Xbox 360 and HDTVs was set with a limit on its resolution. I chose this resolution to be 1280 pixels by 720 pixels, more commonly known as 720p. The p stands for progressive, meaning the entire screen is refreshed sixty times a second, as my game is. This is a standard HD resolution, one supported by the Xbox 360, and is also able to scale up to 1920x1080, or 1080p.

* 1. **Level Data**

The levels of the game are created by reading in values from an XML file. Ideally these files would be turned into objects by XNA, but because I was unaware of this feature until after I already had implemented the levels, and the amount of work required to go back and make changes to something that was already working seemed to be much, I left the levels as they are. The elements of the level file are detailed here.

* Level Number: The number assigned to the current level. For this project values 1-15 are used.
* Level Title: The name of the level.
* Player Location: The location of the player, entered in a Vector2 form.
* Target Locations: The locations of the targets, supporting more than one.
* Bouncy Walls: Provides the origin of the bouncy wall, as well as its width and height.
* Bounce Direction Per Wall: Provides the direction the wall should cause any object to collide with it to bounce in.
* Collide Walls: Provides the origin of the collide wall, as well as its width and height.
* Velocity: The velocity at which the bullet leaving the ship should be traveling.
* Gravity: The direction the gravity should be set at for the start of the level.
* Gravity Value: The coefficient of the gravity in the level.
* Gravity Switchable: Value that tells the level if the player can switch the gravity in the level.
* Switch Limit: Provided the player can switch the gravity, what is the limit to the number of times they can switch the gravity for single shot?
* Gravity Auto Switch: Value that tells the level if the level will automatically switch the gravity.
* Gravity Auto Switch Values: If the level is a gravity auto switch level, these values provide the timers that tell the level how long to use the current gravity and the gravity to use as well.
* Time: The value of the time a player has to complete the level, given in seconds.

Ideally I was aiming to create another program that would make it possible for not only me but others as well to create levels using a GUI interface, but due to time constraints and priorities I was forced to create each level by hand, and test it after creation. The full "code" for all the levels is included with the project source code, but here is a small sample.

<CollideWalls>

<Vector4>

<X>0</X>

<Y>40</Y>

<W>1280</W>

<Z>40</Z>

</Vector4>

</CollideWalls>

This code would create a single CollideWall object with origin at position (0, 40) with a width of 1280 pixels and a height of 40 pixels.

1. **Project Requirements**

Because I was the customer for this project I was able to set the requirements in the beginning of the project. I knew what I wanted the final end result to be, but getting this image from my head into concrete requirements and then being able to follow these requirements was a challenge. Because I was unsure of how long the project would take exactly I decided to separate my requirements into those that I must meet, those that I wanted to meet but they were not vital, and future goals that I would try to get to by the end of this project. This project was coded in C# in Visual Studio 2005.

* 1. **General Requirement Definitions**
1. The game when launched will display the title screen and begin playing the background music.
2. After a set period of time of inaction, the game will transfer to the high score list.
3. After another set period of time, or if the user presses any button from the high score list, transfer back to the title screen.
4. From the title screen if the user presses a button, the game will transfer to the introduction screen which displays the story for the user.
5. From the introduction screen, the game will transfer to the actual game.
6. The user will play the main game and transfer to the next level if the player is able to destroy all the targets the current level contains.
7. While playing the game, various sound effects will play at the correct time.
8. If the user is unable to destroy the targets in the allotted time, the game will progress to the game over screen.
9. If the user is able to able to beat all of the level, the progress to the game won screen, and their score is compared with the other scores to see if theirs is high enough to be a high score.
10. At any time the user can press escape to return to the title screen, or if at the title screen, press escape to exit the program.
11. **In Game Requirement Definitions**
12. In the game the user must be able to rotate their ship by using the keyboard.
13. The user must also be able to shoot bullets out of their ship.
14. If the bullet collides with a collision wall, the bullet will be destroyed.
15. If the bullet bounces with a bounce wall, the bullet will bounce in the correct direction.
16. If the bullet hits a target, the target will be destroyed, and if the target is the last target in the level, the level will progress to the next level.
17. The bullet will be affected by the current gravity of the level.
18. If the user is able to change the gravity, they can do so using the keyboard.

The above are the requirements for the general system, as well as the requirement for the game portion of the project. Below are optional requirements I was hoping to achieve, but was unable to do so.

1. **Optional Requirement Definitions**
2. Allow users to create their own levels and put them into my game. The levels will be created by using a graphical program to place the different objects, and then converted into XML and imported into the project.
3. Improve the current graphics presented in the game. The game will use shaders and other graphical devices to make the game look much better than it does now.
4. When any two objects collide, cause particles to shoot off. This is similar to the optional graphical improvement, but in a different category because this requirement involves adding new things, while the other involves changing already existing things.
5. **System Specifications**

Originally the game was going to be running on an Xbox 360 so requirements would not be an issue. Due to constraints though, the final project will only run on PCs. The PC must have three things installed. These are the .Net Framework 2.0 and .Net Framework 2.0 SP1, Direct X 9.0c, and the XNA Framework Redistributable 2.0. Most users will already have the .Net Framework 2.0 and Direct X 9.0c installed, so the only thing needed to run the game would be to download the small XNA Framework file and install it. Once all of those things are installed, the game may be played.

1. **System Design**

The program is state driven. Each time the XNA Framework calls for an update, sixty times a second, whatever the current screen is updated and then rendered. If the screen needs to transfer, the correct state is reached and the game continues. Below is the state machine that the game follows and the transitions that cause it change states. Not shown is that any state can reach the title screen by pressing escape, and that the title screen can exit the program by pressing escape.

**Game State Machine**



While everything displayed in the game is contained in a screen class object, there are two different main types. They are the in game screens, which contain player objects, walls, targets, and other things. The other type of screens are the screens that just display text like the title screen and high score screen. Below are examples of each type of screen

**Sample High Score Screen**



**Sample In Game Screen**



* 1. **Data Structures**

Each screen that is displayed in the game is related to the states in the state machine. The screens themselves cause their content to be displayed, and also determine if they need to transition to a different screen or not. Each screen contains an update method as well as a render method. The GameScreen class is the screen displayed when the actual game is being played and causes the game to update and render each time the screen is updated and rendered. Levels are created at the beginning of the game and loaded into memory. Once one level is finished, the next level is made active and the game updates that level. Below is a more detailed list of several of the prominent classes. The classes listed are classes that I created, and not classes provided by the XNA Framework.

* Screen.cs : Base class for all other screens. Has the ability to store a texture and render it if needed. All of the non game screen classes use this to render their background, and then they can display their own information in front of the background.
* LevelBuilder.cs : This class is what is able to take the XML file and translate it into data structures that the level class will be able digest and create levels from.
* Level.cs : Each level contains the ship location, target locations, walls, gravity, and anything else the game needs to know to play the level out correctly. The level when updated checks to see if there are any collisions and if so causes the bullet to do the appropriate thing.
* DrawnObject.cs : All of the objects drawn in a level are drawn objects. These are very similar to screens, but where screens drawn the texture at the same location each time, (0, 0), drawn objects contain a position that can change, and a bounding box that allows the level to check for collisions.
* ControlUnit.cs : This is the class that manages the keyboard entry for the game.

There are other classes as well, but these are the basic ones that make up the bulk of my project. There are several classes provided by the XNA Framework that make it easier to manage data. These include classes like Vector4 and Texture2D. The XNA Framework also provides methods to load in images as textures, and load in specially packaged wave files for playing.

* 1. **Algorithms**

Most of the work that would have been handled by algorithms in other programs is handled by XNA in my program. The only algorithms that my project includes are the collision detection between the bullet and any other object and the effects of gravity on the bullet.

Collision detection goes through each wall and target in the level and sees if the bullets bounding box collides with the wall or targets bounding box. If the bullet is colliding with a collision wall the bullet is destroyed. If the bullet is colliding with the target, both are destroyed, and the level is over if that was the last target. If the bullet collides with a bouncing wall, more logic is required. First the direction the bullet should be traveling after it finishes bouncing is found, and then the bullet is marked as bouncing either up and down, or left to right. This is due to a bug I found where a bullet can become stuck inside a wall and continue to bounce back and forth. After the bullet is marked as bouncing, it is moved outside of the wall in the correct direction. While this doesn't cause perfect bouncing, it behaves well enough for my game. If I had additional time, I would refactor the entire way that bullet interacts with bouncing walls.

Gravity affecting the bullet is handled within the bullet. When the bullet is created it gets the gravity coefficient of the level. The bullet is passed the current gravity direction at each update. If the gravity direction has not changed, the velocity of the bullet is altered in the direction of the gravity, with the amount altered increasing as long as the gravity direction does not change direction. If the gravity has changed direction the time is reset to 0 and the bullet begins to experience new forces in that direction, but continues to travel along its previous path at the beginning.

1. **Development Process**

Because I had little previous experience with C# and no experience working with XNA, to begin with the project I spent time working on several small prototypes to get my feet wet and to understand how exactly XNA works. After doing a few prototypes I began the planning for my project. I wanted to make sure that I had mostly everything planned out, at least in rough draft form so that I would be able to implement it in a prototype and then test and improve from that.

Once the planning was over I started to begin the creation of the engine. For testing of the engine I first hard coded a single level and used very basic graphics for walls and the ship. Once I had the game engine worked out and the level was displaying correctly, I moved from a hard coded level to my XML levels. This caused issues to begin with but after I got the first level acting correctly, I was able to look at my plans and create the other 14 levels.

After I had created all of the levels I ran into a few bugs. The biggest was the bug described earlier when a bullet could be contained inside a bouncy wall and not be able to ever exit. After this bug was fixed there were only a few other things I ran into. When I began to track high scores, another thing I saw was that my updated scores were displaying correctly, but not updating the text file containing the scores. XNA stores a local copy of all the files you are working on so the changes were being made and stored, but I was looking at the wrong file.

After all of the bugs had been quashed I began to work on improving the graphics by making new graphics for all of the assets. The only image that I didn't upgrade was that of the bullet. I also created sounds using an 8-bit sound generator for the sound effects, and a program call Korg DS-10 to create the background sound.

As for testing, after any change I made sure to run the game and make sure that things were still working as expected. I also played each level many times to make sure that they were not only solvable, but also was not too hard to finish. Again because I was the customer for this project I had to make sure that I was able to not only meet my requirements, but also to make sure that the end project was working properly, and that the game isn't too easy or too hard.

Once the program was completed I had to test to make sure that each state was reachable with the correct conditions. This included winning with a low score, winning with a high score, losing, and taking no action at all. After I had ensured that each state was reachable, and all things were working correctly, the program was finished.

* 1. **Work Breakdown**

Because I was a novice in using C#, and had never spent any time with XNA I did spent quite a bit of time in research and doing prototypes. Once that was done, most of my time was spent writing code and testing it. Very little time was spent creating the visual and audio aspects of my program. On the next page is a chart showing the schedule breakdown for my project.

Gantt Chart

1. **Results**

Overall the results of the program do match the requirements that I set for it when I began this project. The game works as expected, has the required number of levels, can pose a challenge to a new player, and also is able to track their high scores to offer more reward for playing more than once. The program also has a sound effect engine that plays the sounds required of it, and displays the graphics of a level.

One thing that I wasn’t able to do that I wish I would have had time for was to get a level editor up and working to enable people to create their own levels. As of now they can edit the XML files included with the program, but without knowing exactly how to setup a level, some might be confused. Another shortcoming of this project is that the current algorithm for the bouncing of the bullets and the algorithm affecting the path the bullet follows, while working, is not very good. The current setup makes sloped walls impossible to implement, so if I had time that would be something that gets refactored.

1. **Conclusion**

This project was a lot of fun to work on. I was able to spend time learning a new language as well as a new framework. XNA was quite a surprise to me because of how powerful it is, and yet how easy it was for me to pickup and get done what I wanted. I would suggest XNA to anyone else who might be thinking about making their own video game because of its ease and power. I didn’t use any 3-d assets as all, but the XNA Framework has no problem doing 2-d or 3-d games.

This project also was a great teaching tool. Making requirements and then trying to not only stay on schedule, but meet all the requirements as well can be a challenge. If I were going to do this project again, I would shoot for something a little bit easier, because while now I could get what I have done now quickly, being new to some aspects of C# and new to XNA caused me several issues that took extra time to work out.