Final Design Document

Team  
n00bs

Project  
Got Balls?

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hung-Chin Chang</td>
<td><a href="mailto:hchang@cs.utah.edu">hchang@cs.utah.edu</a></td>
<td>583-4475 (H)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>918-7995 (M)</td>
</tr>
<tr>
<td>Jeremy R. Wright</td>
<td><a href="mailto:jrwright@cs.utah.edu">jrwright@cs.utah.edu</a></td>
<td>486-3430 (H)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>264-1001 (W) x230</td>
</tr>
<tr>
<td>Bryan Witkowski</td>
<td><a href="mailto:bthomas@cs.utah.edu">bthomas@cs.utah.edu</a></td>
<td>484-0592 (H)</td>
</tr>
</tbody>
</table>

Project Description

*Got Balls?* is a revolutionary and evolutionary twist on the classic video game *Marble Madness*. You, the player, will once again take on the epic role of the lone, heroic marble, pitted against the evil machinations of a world committed to destroying you.

The game consists of both a single player and multiplayer mode. The main objective in either version consists of reaching a goal somewhere in the level and staying on it long enough to accumulate points before time runs out. In single player mode, simply traversing the level and reaching the goal counts enough for a win. In multiplayer, you must gain more points than your opponent in order to depart victorious.

Like most modern games, the players and levels are fully 3D modeled. The mouse and keyboard are used to control the marble and change its view. A complete graphics, physics, and collision detection engine was implemented to make the player’s world and movement look as realistic as possible. Networking was added to enable one-on-one play, and sound and music were the final touches.

System Components

We designed our game on and for a Win32 platform. To develop our project, we used Visual C++ .NET exclusively both in the computer lab and at home. To render our graphics scenes, OpenGL was the library of choice. For user input and sound, we used the DirectX SDK libraries. To implement networking, we used WinSock API calls and the UDP transfer protocol.
Because our game uses high-resolution images, textures, and rendering, we recommend at least a later-generation Pentium/Athlon computer with perhaps 128MB of memory and a GeForce or similar 3D video card. The CS lab computers were quite sufficient for our needs thanks to their recent updating. Our program consists of the executable GotBalls.exe as well as several required DLLs, all submitted with our final code. We also submitted the necessary textures, levels, and sounds loaded by the program in their separate subdirectories.

**System Capabilities**

- **3D Graphics Engine** – Contains objects which represent vertices, polygons, balls, eye position, lighting, levels, and textures. The engine maintains its own internal state, and renders each scene using OpenGL calls. It is responsible for displaying everything shown to the user, including the opening game menu and all player notices.

- **User Input** – Accepts keyboard and mouse input, which is then translated to a force on the player’s ball or a rotating of the player’s view. In addition, mouse and keyboard input is used in the main game menu to launch a new game.

- **Networking** – Connects two separate computers both running the game so that their game state can be passed back and forth and synchronized.

- **Sound & Music** – Enhances the game play by triggering sound events when balls collide with the level or with each other. The background music also heightens the emotional state of the player.

- **Physics** – Simulates the changes that occur in the game based on standard laws of physics, such as gravity and momentum. This includes accurately modifying each ball’s position and velocity as it interacts with the level and other balls.

- **Game Engine** – Controls everything from behind the scenes. The game engine receives input from the input and networking engines and uses the physics engine to simulate each new step of the game. The current game and player state are also maintained here. After each simulation step, the new game state is output to the user via the graphics and sound engines.

**Coolness Factor**

We think the actual game idea is rather novel. Although inspired by a few classic games, we had never seen anything quite like our project before. The idea is simple, but in practice it is definitely fun to move a simple object like a marble around a 3D world. It is even more enjoyable to collide at full speed with your opponent and send him careening into oblivion.
Some of the visual effects turned out really well, especially the bump mapping and dynamic lighting. Our single-player level not only looks cool, but it is also quite challenging. The level design definitely plays a big part in making the game play enjoyable.

**Individual Contributions**

<table>
<thead>
<tr>
<th>Chang (1/3)</th>
<th>Jeremy (1/3)</th>
<th>Bryan (1/3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Networking</td>
<td>• User Input</td>
<td>• Graphics Engine</td>
</tr>
<tr>
<td>• Sound &amp; Music</td>
<td>• Physics</td>
<td>• Collision Detection</td>
</tr>
<tr>
<td></td>
<td>• Game Engine</td>
<td></td>
</tr>
</tbody>
</table>

**Accomplishments**

We all feel very pleased with what we achieved. As the end approached, we were definitely behind schedule, and the networking appeared to be an almost insurmountable challenge. Consequently, when we finally managed to implement multiplayer support, that felt like one of our biggest accomplishments.

Another one of our biggest goals was to get experience actually working on developing a game of this complexity. To actually even begin coding our respective modules, we had to spend dozens of hours looking through documentation, tutorials, theses, and textbooks. Just the amount of knowledge we gained made this whole project worthwhile.

Perhaps the greatest satisfaction, however, was to see people sit down and enjoy playing the game we worked so hard on creating.

**Team Experience**

Working as a team definitely had its challenges. We all had widely different schedules with heavy class and workloads, so it was usually tough to meet more than weekly. Often due to the language barrier, it was even a challenge to communicate with each other. Our progress was hampered by some misunderstandings on how all of our separate modules needed to communicate with each other. It was difficult to help each other when one person ran into bugs or unexplainable errors with his code.

Fortunately though we formed our team based on our respective and complimentary strengths. We counted on each other to be sufficiently capable in our own areas of expertise, and for the most part we were. In the end, we each managed to fulfill all of our designated responsibilities.
Large Project Experience

As mentioned previously, the experience of working on a project of this magnitude and scope was something that none of us had participated in previously, at least not in a school setting. Working on a game is particularly complex because it involves so many varied aspects of computer science. Having a single large project due at the end of the semester is definitely a change of pace from the usual style of homework, and it required a lot more discipline on our parts.

Meeting regularly with the professor was of course beneficial, as was having a prototype day halfway through the semester. Despite that, the bulk of the responsibility for keeping up with our schedule fell on our shoulders. There were occasions when we didn’t always manage our time perfectly, but for the most part we spent a proportionate amount of time on our project each week.

This project was definitely a valuable experience because the projects we work on in our future jobs will require the same team and time-management skills we developed here.

Time Machine

Hindsight, as they say, is 20/20. As usual, the proverb is correct. There are a lot of obstacles we faced that could have been avoided.

First off, we probably set our expectations too high, especially considering that we would be coding mostly everything from scratch. It was always an option to use an existing graphics or game engine, but we wanted the experience of coding things ourselves. Estimating what one can program over a long period of time is tough at best, so we should have been more conservative in our expectations.

Secondly, we should have met much more frequently early in the semester. A substantial amount of time and code was wasted as a result of misunderstood or unclear specifications. Had we met more often, those problems would have been spotted and rectified much more quickly.

Last but not least, we probably would have done differently would be to spend more time in the design stage together. We each designed our own specific modules reasonably well, but oftentimes the best or easiest way to program your module may not be the best choice for fitting your module in with everyone else’s code.

Despite our failures though, our final accomplishments hopefully speak for themselves. We are proud of what we created, and that perhaps is what is most important.