Natural User Interface and Virtual Reality Integration in Video Games

Last Revised: April 1, 2015

Alexandre Wimmers	Mario Yepez	Timothy Tong	Valerie Gadjali
adwimmers@ucdavis.edu	myepez@ucdavis.edu	tktong@ucdavis.edu	vgadjali@ucdavis.edu

1 Summary

Castle Defense aims to provide the user with a fun and enjoyable gaming experience for everyone. The game makes use of Virtual Reality and a Natural User Interface to provide this experience by using emerging technologies such as the Oculus Rift Dk2 and Leapmotion. Should any problems or questions arise, feel free to contact us at `castledefensegame@gmail.com`

2 Resources Required

2.1 Hardware Requirements

Leap Motion	`https://leapmotion.com/`
Oculus Rift DK2	`https://oculus.com/dk2`

2.2 Software Requirements

Leap Motion SDK	2.2.2.24469
Oculus Rift SDK	0.4.4 Beta
Unity	4.6.1
Unity Test Tools	1.5

3 Build Instructions

1. Checkout the Bitbucket repo: `git@bitbucket.org:ecs193/project.git`
2. Open the project in Unity
3. Optimize the build for the Oculus Rift by going to Prefences→Oculus Vr and checking 'Optimize Builds for Rift'

4. Build the game using the default settings provided by Unity.

5. Run *CastleDefense_DirectToRift.exe*.

4 Issue Reporting Procedure

1. File an email report to `castledefensegame@gmail.com` with subject line "[ECS 193 — Issue]".

2. Provide a brief description of the issue.

3. If possible, provide screenshots and steps to reproduce the error.

5 Functional Testing

5.1 Unit Testing

Class/Entity	Method	Assertion	Description
Castle	Init()	CurrentHealth == MaxHealth	Assert that the castle starts with max health.
	LoseHealth()	CurrentHealth == (MaxHealth - Δ)	Assert that the castle's health depletes and depletes by a correct amount.
	GameOver()	CurrentHealth == 0 GameOver.GUI == Activated	Assert that the castle health is depleted and that the end game GUI has activated.
Enemy	Init()	CurrentHealth == MaxHealth Target == Castle	Assert that the enemy spawns with max health and that the enemy target is the player castle.
	LoseHealth()	CurrentHealth == (MaxHealth - Δ)	Assert that the castle's health depletes and depletes by a correct amount.
	Death()	CurrentHealth == 0 Speed == 0 Verify(DeathAnimation).Called(1) Verify(this).deleted	Assert that the enemy health is 0. Assert that the enemy is no longer moving and that the death animation was called. Then assert that the enemy entity was deleted.
Enemy Manager	Spawn()	SpawnPoints[] != NULL SpawnType != NULL Verify(Enemy.Spawn()).Called(1)	Assert that spawn points and spawn type is not NULL. Assert that an enemy was spawned or instantiated.
	Spawn()	canSpawn == FALSE Verify(Enemy.Spawn()).Called(0)	Assert that if the enemy manager is disabled that an enemy is not instantiated.

5.2 Integration Testing

Class/Entity	Method	Assertion	Description
Castle	GameOver()	CurrentHealth == 0 GameOver.GUI == Activated EnemyManager.canSpawn == FALSE Enemy.disabled == TRUE	Assert the same requirements as unit testing. Also assert that the enemy manager spawn and enemies are disabled.
Enemy	Death()	CurrentHealth == 0 Speed == 0 Verify(DeathAnimation).Called(1) Verify(this).deleted Player.Score++	Assert the same requirements as unit testing. Also assert that the player's score increases.

6 Non-Functional Testing

6.1 Security

Not applicable. This product does not contain client sensitive data. The issue of potential hacks into the game is a considerable issue. All Unity games execute through the Unity platform similarly to how Java executes in the Java Virtual Machine. By default, there is "natural" security through this platform. Therefore, if our game was to be compromised, then the issue persists to the entire Unity platform, which falls under the jurisdiction of the Unity Security Team.

6.2 Fault Tolerance

Not applicable. The game does not perform any form of persistence.

6.3 Performance Testing

Build and run the game. Use the frame-per-second (FPS) counter to poll an average FPS. An average FPS below 30 is unacceptable.