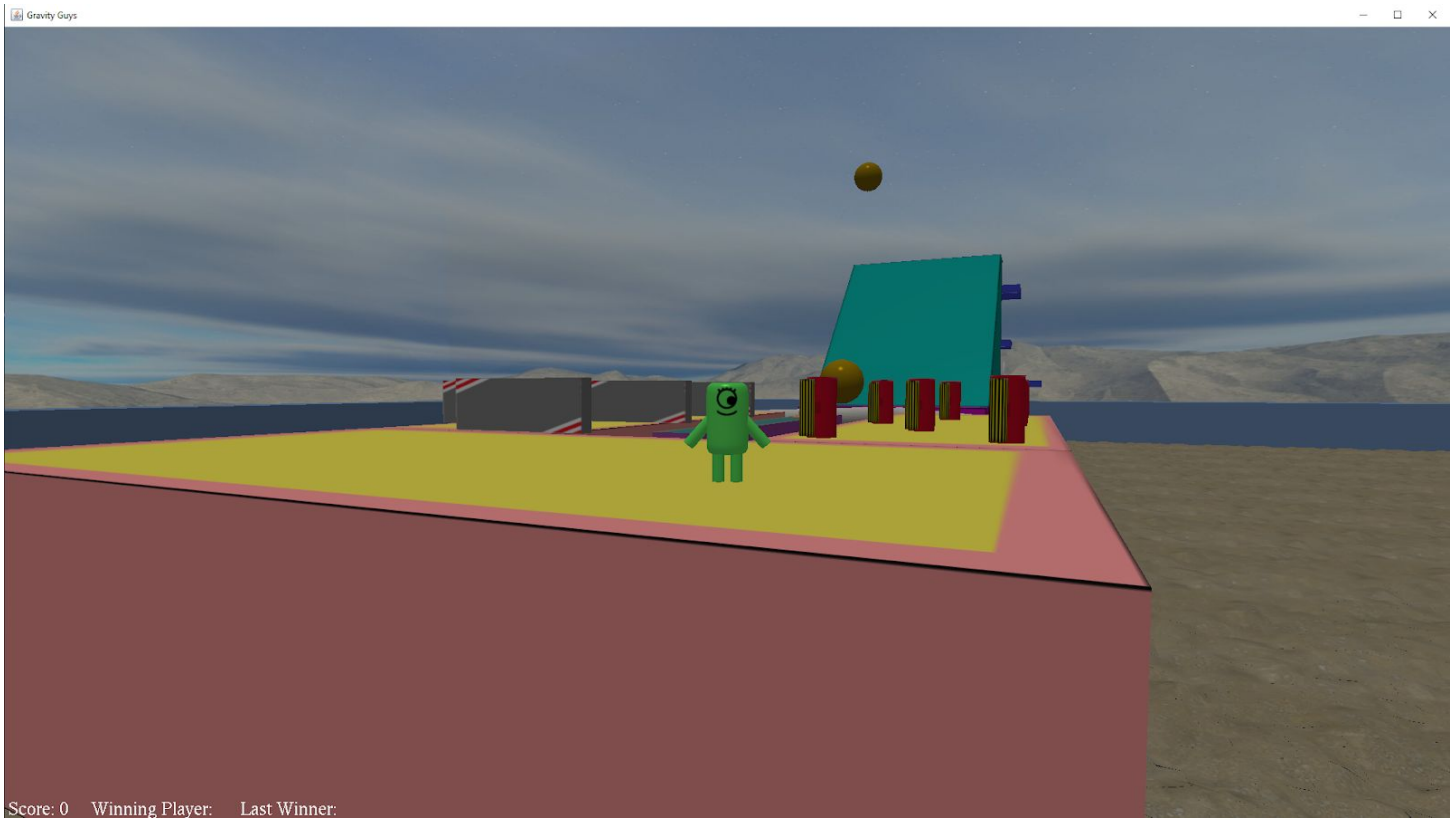
Sacramento State University

Gravity Guys - Player Guide

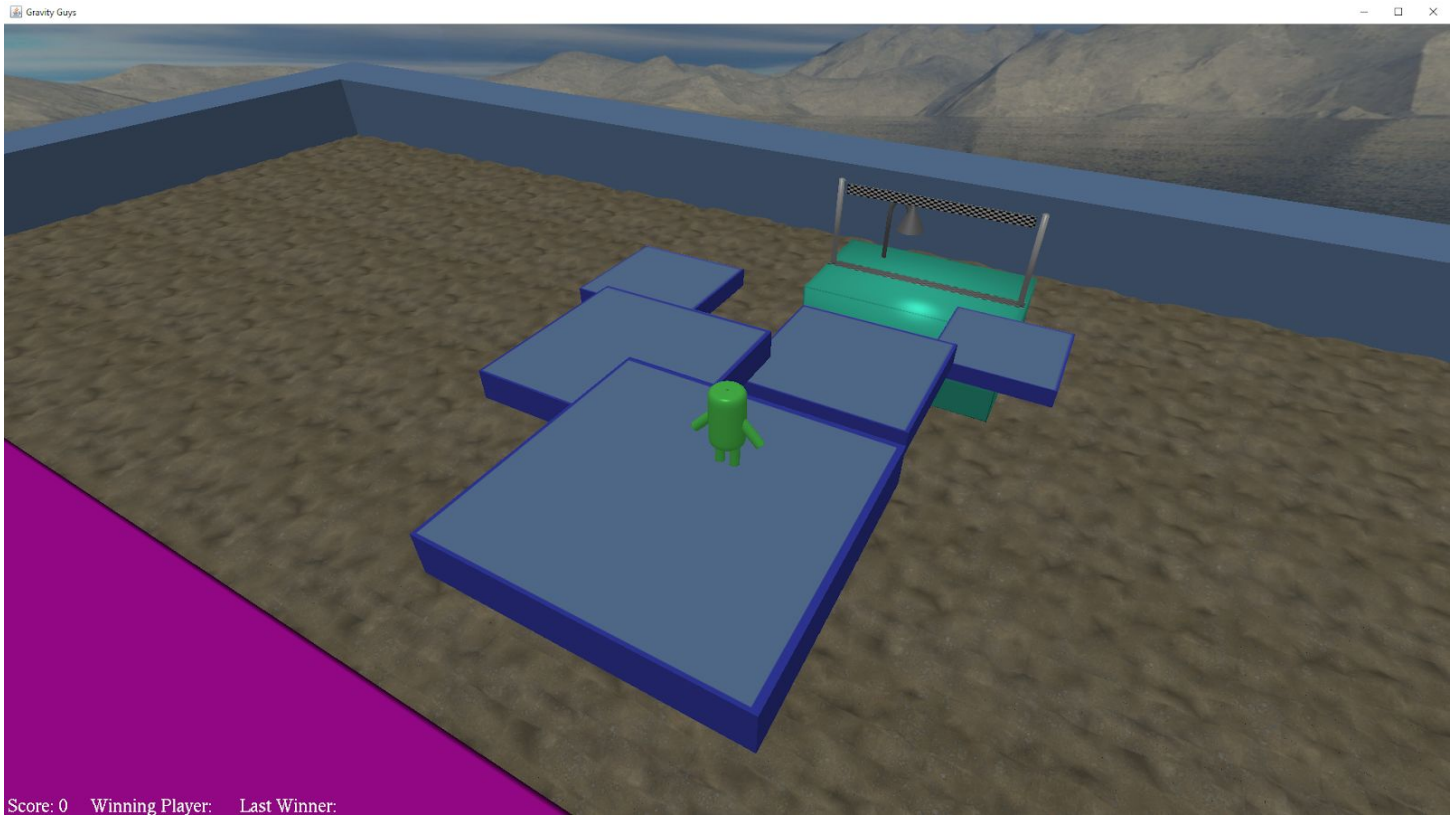
December 16, 2020

Quinn Roemer and Josh Hutton
CSC 165-02
Dr. Scott Gordon

2 - Game Screenshots:



(Game starting platform with obstacles)



(Moving platforms, that the player must jump onto to travel to the finish line)

3 - Compiling and Running the Game from the Command Window:

The game can be compiled and ran from the command window by opening a CMD window inside the game directory and running the following commands:

- javac a3*.java [IP address] [port number]
- java -Dsun.java2d.d3d=false -Dsun.java2d.uiScale=1 a3.MyGame

Alternatively, you can just use the provided batch files to perform this operation. Run compile.bat and then runGame.bat. The latter batch file can be modified with the game server's IP and port number.

In order to establish networking, the server must be run, this can be done by opening a CMD window inside the game directory and running the following commands:

- javac myServer*.java
- java myServer.NetworkingServer [port number]

Alternatively, you can just use the provided batch files to perform this operation. Run compile.bat and then runServer.bat. The latter batch file can be modified with the desired game server's port number.

If you wish to connect a client to the server, you must include the server's IP and port number either in the command line or inside the batch file used to run the client. If the game fails to parse the provided IP address and port number, it will default to port 89, and use your local IP address.

Please note, if you run into the following error "Error: A JNI error has occurred, please check your installation and try again" delete the .class files in the *a3*, *myGameEngine*, and *myServer* folders. Then recompile the project and try running again. This error occurs due to compilation on a different Java version.

4 - Device requirements:

Our game can be played using a keyboard/mouse combo and/or a gamepad. There are no other special requirements.

5 - How to Play Our Game:

The goal of our game is to reach the finish line. However, in your way are obstacles such as moving walls, rotating flails, and falling balls (and much more). Avoid these obstacles and navigate to the finish line. If at any time you fall off the platforms you can reset yourself back to the beginning using the reset player button found under our game controls. If the player is not connected to the server, reaching the finish line will reward a single point. After 10 seconds the game will reset (keeping your score) allowing you to go again. If the player is connected to the server, the first player to reach the finish line (assuming multiple players) will be awarded two points. After 10 seconds the game will reset like before. However, if other players are able to reach the finish line within the 10 second timer, they will be awarded a single point. After the game resets, the clients will display the current winner and the last player who won along with your current score.

6 - Game Controls (Keyboard/Mouse | Controller):

Move player forward	W	Y axis (left stick)
Move player backward	S	Y axis (left stick)
Move player left	A	X Axis (left stick)
Mover player right	D	X Axis (left stick)
Yaw player left	Q	Z axis (left and right trigger)
Yaw player right	E	Z axis (left and right trigger)
Move camera azimuth	Mouse X axis	RX axis (right stick)
Move camera elevation	Mouse Y axis	RY axis (right stick)
Adjust camera radius	Scroll Wheel	POV hat (forward/backward)
Jump	Space	Button 1 (A)
Reset player	R	Button 6 (right button)
Toggle light	F	Button 2 (B)

7 - Scripting:

Scripting is used to initialize variables for game variables, lights, and movement. Our game has the functionality to support updating the scripts during runtime through the *UpdateGameVariables* and *ScriptManager* classes. However, any updates to the *gameVariables* script will break the physics world. As a result, it is recommended that you restart the game after doing so. This does not apply to updates in the *movementInfo* script. You can find our scripts in the *Scripts* folder.

8 - Network protocol:

Our networking protocol is based on the protocol provided in the network handouts. However with many additional added features and functionality. For example, the game server at all times stores the location and orientation of all clients. This limits the "back and forth" communication required when a client wants updated information for a ghost. Also, our server is capable of detecting clients that may have closed without properly informing the server through the *DeadClient* class. Some examples of new commands that have been added to our protocol follow: NPCROT (send by a client to tell the server the NPC rotated), JUMP (sent by a client to inform the server its jumping), STOPJUMP (sent by a client to inform the server it has stopped jumping), KEEPALIVE (sent by a client every 10 seconds to tell the server its is active) and WIN (sent by a client to tell the server its reached the finish line). Many more commands have also been added server side.

9 - Game description:

1. **Game Genre**: Obstacle course/platformer
2. **Game Theme:** Geometric shapes with vibrant coloring
3. **Game Dimensionality:** 3D
4. **Game Activities:** Race to the finish line, obstacle avoidance, and platforming

10 - Game requirements:

- **External Models:** Josh created the player model, lamp model, and various platform models. Quinn created the fan model, and flail model. These models were UV-Unwrapped and include custom textures. All the models in our game were either included in RAGE, or made by one of us.

- **Networked multiplayer:** Our game is playable in either single player mode or multiplayer. When connected to the server, a maximum of 8 players can play. Other players are represented via "ghosts" in each client's game world. A player is able to select their desired color through the dialog box presented for selecting window size. If this color is available the server will assign the color to that specific client. However, if it is not available the server will automatically assign the player a new color that is available.

- **Scripting:** Scripting is used for initializing game variables, lights, and movement. Dynamic updates are available on all script files with one caveat. Any modification to the *gameVariables* script will break the physics world. After updating this file it is recommended to restart the game. All scripting was done in JavaScript.

- **Skybox and terrain:** The sky and mountains in the distance are a skybox, and the sand in the sandbox below the obstacle course is terrain. If the player falls into the sandbox, their vertical position will be updated according to the height map. However, iif the player is still on the obstacle course, they are unaffected.

- **Lights**: Ambient and directional light are used in our game. On the finish line platform, there is a lamp that is off by default. Once the player is close enough they can hit the button listed under game controls to toggle the lamp on and off.

- **3D sound:** There are 3 sound effects in our game. These are walking, jumping, and a blowing sound effect. The sources for the walking and jumping sounds are the players, and the blowing sound's source is the fan. These sounds are implemented with RAGE's audio package.

- **HUD:** The HUD tells the player their current score, the current player with the highest score, and the winner of the previous round. We used the built-in HUD provided by RAGE.

- **Hierarchical scenegraph:** Many nodes are organized hierarchically in our scenegraph, however a clear example is our levelOne node. This node is the parent to all of the node objects that make up the obstacle course itself.

- **Animation:** The player model has a looping walking animation and a jump animation for its respective actions.

- **NPC:** The fan on its own platform near the start of our game is an NPC. The NPC patrols the platform back and forth until a player gets within range, then it blows the player away until it is out of range and goes back to patrolling.

- **Physics:** All player movement is done through the physics engine. Being blown away or colliding with obstacles is also done through the physics engine withholding the flails. Most objects in our world are modelled in the physics engine allowing for players to interact properly with them. We used RAGE's built in physics package.

11 - Missing Requirements:

We were able to implement all the requirements successfully.

12 - Additional Modifications:

- The Orbit Camera pitches up when it touches the ground planet to enable the player to look up.
- The game server stores some limited information on clients to facilitate faster communication between clients.
- The game server is multithreaded and performs NPC control, dead client detection, ball spawning, and game reset in separate threads.
- While on the platforms after the wedge the player moves with the platform if on one.
- When not connected to a server, the client takes command of the NPC, ball spawning, and game reset.
- When the player moves beyond the sandbox the player is automatically ported back to the start of the game.
- The client is capable of connecting to a server even if the server was started after the client. Provided, of course, the IP and port number provided to the client were correct.

13 - Contributions of each team member:

The skybox, scripting, networking, HUD, and NPC/AI was implemented by Quinn Roemer. Player models, textures, animations, lights, sound, and animation was implemented by Josh Hutton. Terrain and Physics were equally shared between us.

14 - List of assets we created:

Textures:	
bluePlayer.png	Created by Josh Hutton in Blender and GIMP
brownPlayer.png	
greenPlayer.png	
orangePlayer.png	
pinkPlayer.png	
purplePlayer.png	
redPlayer.png	
yellowPlayer.png	
finishPlatform.png	
groundPlatform.png	
lamp.png	
platform.png	
wedge.png	
wishbone.png	

ball.jpg	Created by Quinn Roemer in Blender and Microsoft Paint
cylinder.png	
fanBase.png	
fanBlades.png	
npcPlatform.png	
physicsPlatformTexture.png	
pillar.png	
redCube.png	
sandBoxWalls.png	

Meshes:

finishPlatform.obj	Created by Josh Hutton in Blender
groundPlatform.obj	
lamp.obj	
platform.obj	
player.rkm	
wedge.obj	
wishbone.obj	

customCube.obj	Created by Quinn Roemer in Blender
cylinder.obj	
fanBase.obj	
fanBlades.obj	

Materials:	
finishPlatform.mtl	Created by Josh Hutton in Blender
groundPlatform.mtl	
lamp.mtl	
platform.mtl	
wedge.mtl	
wishbone.mtl	
customCube.mtl	Created by Quinn Roemer in Blender
cylinder.mtl	
fanBlades.mtl	
fanBase.mtl	

Animations:	
jump.rka	Created by Josh in Blender
newWalk.rka	

Skeletons:	
player.rks	Created by Josh in Blender

Sounds:	
jump.wav	Created by Josh Hutton in Audacity
wind.wav	Created by Quinn Roemer in Microsoft Voice Recorder

15 - List of Sourced Assets:

Textures:	
default.png	Provided in RAGE
hexagons.jpeg	
sand.jpg	Sourced from cc0textures.com and distributed/used under CC0 1.0 Universal allowing for free use, adaptation, and sharing for any purpose.
tileableHeightMap.png	Sourced from opengameart.org and distributed/used under CC0 1.0 Universal allowing for free use, adaptation, and sharing for any purpose.
tileableNormal.png	

Meshes:	
cube.obj	Provided in RAGE
sphere.obj	

Materials:	
cube.mtl	Provided in RAGE
default.mtl	
sphere.mtl	

Skybox:	
calm_sea/back./jpeg	Provided in RAGE
calm_sea/bottom.jpeg	
calm_sea/front.jpeg	
calm_sea/left.jpeg	
calm_sea/right.jpeg	
calm_sea/top.jpeg	

Sounds:

walk.wav	Sourced from https://www.fesliyanstudios.com and distributed/used in the context of "video games" with the authors permission **File originally named** sneaker-shoe-on-concrete-floor-fast-pace-1-www.FesliyanStudios.com.mp3

16 - Riverside 5029 Machines:

Tested and working on ECS-WARCRAFT (Client and Server) & ECS-ASTEROIDS (Client).

Note: If a client has difficulty connecting to the server, recompile that client and try again.