CPS511 Assignment 3

Robot Attack

(Worth 20 percent of your mark)

Due Date: Monday, December 2, 11:59 pm

(Note: You may submit your assignment until Sunday, December 8, 11:59 pm, without penalty. No assignments will be accepted after December 8, 2024.)

In this program, you will combine ideas/code from Assignment 1 and Assignment 2 to create a 3D "Star Wars Planet Hoth" style game of walker bots attacking an interactive defensive cannon. This final programming assignment will increase your knowledge of all the concepts we have learned in the course. **You must do this assignment individually or in a group of two. Do not attempt to find source code on the web for this assignment. It will not help you, and you risk extremely serious consequences.** You may reuse code from *your* Assignments 1 & 2**. If you did not get 100% on assignment 1, fix it with the provided feedback or ask another group for their assignment 1 code. INDICATE which group's assignment 1 code you use in the readme file.** Begin designing and programming early! Start by reading this description carefully.

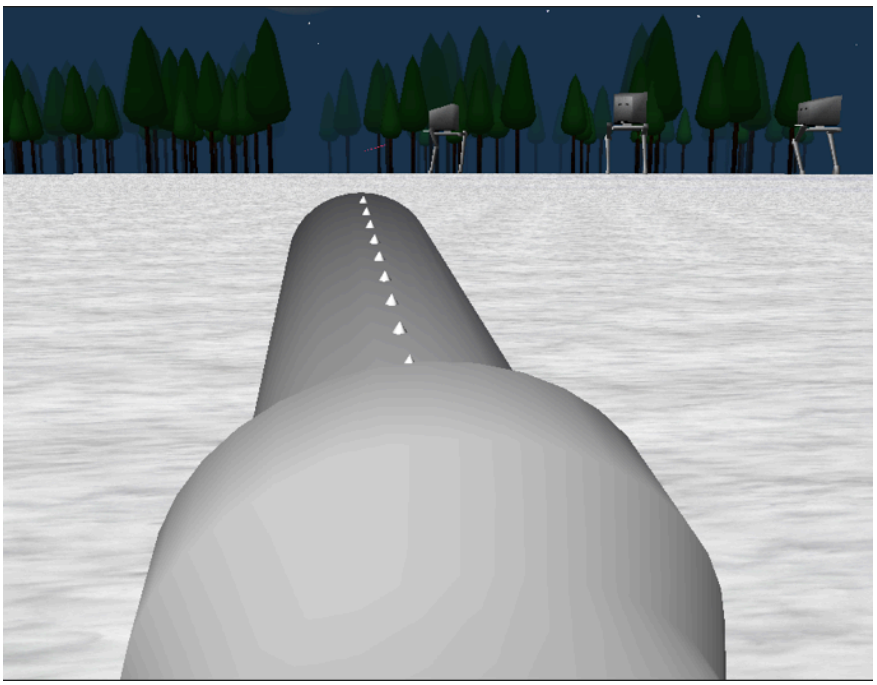

Figure 1: Walkers Attacking Defensive Cannon!

Program Functionality Requirements

1) Enemy Walker Bots Attacking

a. Use your robot from Assignment 1. Create several instances of your robot. If you prefer, an enemy robot walks along the ground (in a straight line or some other pattern) toward the defensive cannon. **See the example video**. As it walks forward, the robot should fire a projectile toward the defensive cannon (either a regular firing pattern or random). Give the projectiles random accuracy so they don't hit the defensive cannon constantly. **You must extend A1 functionality for full marks and make your robot walk forward**. **It doesn't have to look realistic!** If you cannot get the walking functionality to work or do not want to attempt it, just translate the robots towards the cannon without moving the legs.

b. If an enemy robot is hit by a projectile coming from the defensive cannon, it should disappear. **For bonus marks, animate this disappearance.** For example, the bot could shrink and disappear. Or break apart and then disappear. Use your imagination. In the example video, the robot "collapses". If the robot reaches the end of the ground plane, it should also disappear. Once all robots have been "killed" or have passed the cannon, you may start a new wave of robots or reset the game with a key.

c. Use your shape modeller from assignment 2. Generate **at least 1 custom mesh** and use it as a part of your enemy robots or the defensive cannon. You can load custom mesh parts in a couple of ways in this assignment. You can store the mesh into a file and load the file into data structures in the assignment 3 code. Alternatively, you can cut and paste the mesh vertex array and mesh quad array from assignment 2 (containing all the values) into assignment 3. Finally, you could incorporate enough code from Assignment 2 into Assignment 3 to auto-generate the mesh.

d. You must texture-map all parts of your robots to make them look interesting. If you use **glut-**shape primitives, you cannot texture map them. Either switch to **glu** shape primitives or use your mesh parts to replace them. If you use glut shape primitives (i.e. not texture mapped), you will lose marks. **Make sure you use MODULATE mode when doing texture mapping (i.e., combine lighting + shading with texture mapping). NOTE: you can texture map glut shape primitives using a fragment shader for lighting, shading and texture mapping.**

2) The Defensive Cannon

a. Your cannon should be controllable via the mouse. Left-right movements of the mouse move the cannon; left-right, up-down mouse movements move the cannon up and down. **See the video for an example**. If you cannot get the mouse working, use the arrow keys. As mentioned above, the cannon can fire projectiles at the incoming robots (use the space-bar key). You must texture-map all parts of your defensive cannon to make it look interesting. If you use **glut-**shape

primitives, you cannot texture map them. Either switch to **glu** shape primitives or use your custom mesh parts to replace them. If you use glut shape primitives (i.e. not texture mapped), you will lose marks. See the comment above regarding using a shader to texture map glut shape primitives.

b. The viewer should be positioned just above and towards the back of the cannon (see the figure) so that you can see part of the cannon and the ground and incoming robots.

c. If a projectile hits the defensive cannon, it should stop working. The cannon should animate to a "broken state." For example, the cannon barrel could animate down to indicate it no longer functions. Bonus marks will be given for more complex animations of a broken state or using a special effect to explode the cannon somehow. **See the Hints below to detect whether a projectile has hit the defensive cannon.**

d. **Use a key to restart the game so the user can continue playing.**

3) **Final Requirement:**

To be eligible for full marks on this assignment, you must use a GPU vertex shader and a GPU fragment shader to render your scene using lighting, shading, and texture mapping. I will post a link or example program to help you write the shaders.

Hints

- You could use several techniques to check collisions between a projectile and a robot or a projectile and the defensive cannon. A simple technique is to compute the distance between the projectile and the center of the robot (or cannon) – if this distance falls below a threshold (i.e. is "close"), then a collision has occurred. Alternatively, for a robot and/or the cannon, you could construct a rectangular bounding box (i.e. compute the minimum and maximum x, y, and z values of a robot when centered in its own coordinate system – similarly for the defensive cannon). Check to see if a projectile's current (x, y, z) position is inside this bounding box (i.e., simply check the projectile x component against the minimum x and maximum x range of the bounding box. Repeat for y and z.). You will need to calculate the current minimum x and maximum x (and y, z) based on the current center of the bot.
- **NOTE: you must continually check for collisions each time you render.**

Optional Bonus (1 mark for each part, maximum of 2 marks)

1. Have the defensive cannon fire a laser instead of a projectile [1 mark]
2. Add more complex special effects when a robot is hit and/or when the cannon is hit. For example, you could show pieces flying away, robots breaking into pieces, sparks, explosions, simulated flames/smoke, or other effects. [1 mark]

3. Add levels to your game (and display a score). Each level could have different parameters (e.g., more and/or speedier robots, different weapons, etc.). Add comments to a README file to detail what you have done. [2 marks]

The TA will judge the difficulty of your added functionality when deciding on bonus marks. The maximum bonus marks are 2. I recommend not to work on the bonus code until you have completed 100% of the base program functionality.

Grading (out of 20 marks)

Requirements	Mark Breakdown
At least 2 texture-mapped robots walking forward along the ground and firing projectiles at the defensive cannon. The defensive cannon is texture-mapped. Defensive cannon fires projectiles using the space key. Projectiles are rendered and animated.	10 marks for at least 2 bots 5 marks if simple robot translation with no walking
Successful detection of collisions between defensive cannon projectiles and the robots and projectiles from robots hitting defensive cannon. Robots disappear when hit or some other disabling animation.	2 marks for some kind of collision indicator and/or animation 1 mark if collision detection is purely in code without visual indication for the player.
A defensive cannon hits a projectile and animates to a stop-working state.	1 mark
Mouse-controlled defensive cannon	4 marks for mouse 2 mark if using only keys
Use of custom mesh generated via assignment 2	1 mark
Use of vertex shader and fragment shader to render the scene (lighting and texture mapping)	2 marks 1 mark if there is no texture mapping
Bonus (**You cannot go above 100%**. The bonus compensates for any potential deductions; we encourage you to attempt them!)	Up to 2 marks
Total:	20 marks

Program Submission

Use D2L to submit your assignment. Submit all your source files. You should use C for your program (.c files). You may use C++ (.cpp files). **Do not include executable files!** If your program runs under Windows, include a README file describing how to compile your program. If you want to inform the TA about your program (special features, bonus work, etc.), include this information in program comments and the README file. Include all solution files (e.g. Visual Studio project files) or makefiles (for Linux/Mac).

Include a video of up to 5 minutes demonstrating all of the requirements. Record directly from your desktop using a professional screen recording tool like Zoom rather than a cellphone. Add voice narration throughout the video to explain each requirement as you demonstrate it, ensuring a clear and concise presentation.

It is your responsibility to ensure the TA has enough information so that they can, with little effort, compile and run your program. If the TA has trouble compiling your program, they will have the discretion to deduct marks and/or ask you to compile and run your program in person.