Mars Mission Control Game in WebGL

Report Name	Mars Mission Control Game Project Specification
Author	Samuel David Snowball, User id: sds10
Supervisor	Dr Helen Miles, User id: hem23
Module	CS39440
Degree Scheme	G400 (Computer Science)
Date	February 09, 2017
Revision	1
Status	Final

1. Project description

The project will involve building an interactive mars mission control game, allowing the user to roam about on mars as a rover to complete certain tasks. It will be built using: WebGL, JavaScript and some HTML/CSS. Due to it being built in WebGL it will be played via a browser. I will use resources [1-3] to continue learning from throughout the project.

My motivation for the project is learning how computer graphics works and making something interesting at a reasonably low level. Programming in WebGL will help me gain understanding of what's actually going on to get the game to display on the screen. Learning OpenGL/WebGL is useful as it's flexible and used on various operating systems, rather than Direct3D which is only for windows.

I haven't been able to find a mars rover game built like this before. NASA has published a mars 2D rover game, but mine will be quite different. This is mainly due to mine being in 3D, and also because my game will have more features rather than only avoiding obstacles and detecting water like NASA's version.

I will use the m4.js matrix library [4] to help with the project, as I don't have the knowledge to write matrix operations from scratch.

The project will be open source, licensed under MIT. This means players can change and break the game code as they wish, but due to it being single player I don't feel like this is an issue. The project is currently hosted on GitHub at:
https://github.com/SamuelSnowball/Major-Project

My methodology will be SCRUM, due to scrums flexibility and it being an agile approach. Since I'm not confident in WebGL, I feel an agile approach is best as I can focus on working software early, rather than getting 4 weeks in and realizing I can't code WebGL.

I will have initial requirements/tasks (described in section 2).

I think 1 Sprint per week is adequate, tackling multiple tasks. Monday through to the next.

At the start of each day

- Daily scrum, 15 minute planning for the day ahead, reading over sprint backlog and improving if needed. Shows the probability I will meet sprint goals. In the daily scrum I will go over:
  - What did I do yesterday?
  - What will I do today?
  - Any problems that will stop me from achieving sprint goal?

At the start of each sprint

- Sprint planning at start (1 hour) describing some stories/tasks in more detail before implementation
- What stories can be completed (the sprint goal)
- How will the stories get implemented

At the end of each sprint

- Sprint review, put sprint backlog into product backlog, say what was done, how and why. Then review what stories are best to tackle next.

- Have a sprint retrospective, what could be improved in future. Describe any issues I faced and think of ways to overcome them. What went well in the sprint, and what didn't. Then adapt for the next sprint.

The product backlog (my changelog on GitHub) could act as the required blog. It will say the current requirements, their priorities, and some estimations. It will also include what has been added to the project, and what's left to add.

The current sprint backlog will be inserted into product backlog once the sprint is completed. I will have a bug log for good practice and will use screenshots to give an easy way of seeing my progress. I will aim for a working release after each sprint.

2. Proposed tasks

I will assign priorities to tasks/stories, and complete the most important ones first. Getting main mechanics finished early is much better than doing GUI's or other less important things, as it gives me time to implement mechanics properly.

I will mainly research features as I need to build them, so the research process is likely to go on throughout the project. Researching lots at the beginning wouldn't be very useful as I might not be able to apply the information straight away, as other features are needed first. Working software early is also a priority.

Some general information about the project and game mechanics:

- Focus on first person, perhaps adding in third person later on. A first person view will feel more interactive
- User will interact via keyboard and mouse
- I will need to test and cater for multiple browsers

Tasks (in loose priority order, highest priority first):

- Build terrain, currently using perlin noise - would be cool to build from existing height maps. However with perlin noise I get flexibility of what I want. Adding collision to existing terrain that I haven't generated would be difficult. This terrain building process would probably be ongoing, as I add new areas and such throughout the project.
- Allow user to move around terrain, camera class needed
- Add collision with terrain, rather than flying over it. Possibly connect terrain vertices with quad strip for more realistic collision, rather than existing triangle strip
- Give terrain texture, procedurally generated perhaps
- Add in rocks. I've found a procedural rock generation library I could use to generate awesome rocks. However I would rather try implement rocks myself. I could use a sphere geometry and use perlin noise on the vertices to give a bumpy geometry.
- Start work on GUI's, with a possible (not necessary) minimap, and other GUI's showing the rovers stats and location (perhaps just their coordinates). Commands from mission control would get sent here, the user would then complete them. Example: go to the X,Y,Z quadrant, find and sample X rock.
- Add in various missions for the player to complete, perhaps adding in levelling mechanics. For example once the player has completed x number of missions he can access new areas.

- Add audio
- Add clouds
- Add sunlight, shadows – I'm unsure how to do lighting in WebGL so would require a lot of research, perhaps taking up a whole sprint. Or it might not be feasible to do this at all., just an idea.
- Add initial loading scene, rover jetpacks into the landing site, terrain comes towards player. This wouldn't take long.
- Add water, I imagine this would take a whole sprint in itself but it's something that would be good to add, from a technical and graphical side. I would have to use various tutorials and resources to be able to do this.

I expect many more features will get added to this in future, the main focus of this list was to get the main mechanics down.

3. Project deliverables

- Terrain generation
- User navigating terrain
- User collision with terrain
- Textured terrain/added in rocks
- Some GUI's and the user receiving and completing missions
- Then after this, probably random extra features like clouds and audio

These project deliverables/releases would contain various design and technical information as well. I might end up including multiple of these topics into releases if I get a lot done, rather than 1 release per bullet point.

4. Initial annotated bibliography

[1] Jamie King. 3D Computer Graphics Playlist Using OpenGL. https://www.youtube.com/watch?v=6c1QYZAEP2M&list=PLRwVmtr-pp06qT6ckboaOhnm9FxmzHpbY 2013.

This was the original playlist I learnt OpenGL from, I made the transition to WebGL several weeks ago, and I've noticed few differences.

[2] Website with basic WebGL tutorials. Created by Gregg Tavares. https://webglfundamentals.org/

This site contains a series of tutorials on basic WebGL principles, it's up to date and has good explanations.

[3] A 3D Game using OpenGL and Glut. Created by Youtube user OneGoldenCat. https://www.youtube.com/watch?v=aviL3HX3UEc&t=80s May 12, 2015.
This video also contains a link to the source code of the game, which I've downloaded and could be useful in the future. It has several mechanics that I'm also interested in building, such as realistic collision and game physics.

[4] m4.js library used for handling matrix operations. Created by Gregg Tavares. https://github.com/greggman/webgl-fundamentals/blob/master/webgl/resources/m4.js

Contains more functions than other matrix libraries like the Mozilla Development Networks library I originally used.