IMAT3905 Report

Please write in the boxes below. Expand the boxes as you need to, however this report should not exceed 4 pages without appendices.

Names:	1) Alfie Horton 2) Adam Hubble 3) Jack Moorin ~~4) Joseph HIll~~	P Numbers:	1) P17175393 2) P17175774 3) P17190172 4) ~~P17174751~~
Github Username:	Team Funk	Github Repo URI:	https://github.com/IMAT3905/cw-2019-2020-team-funk.git

Please summarise the functionality of your individual contributions to the game engine:

(name):
Alfie Horton

(Contribution)

- Added physics to the engine.
  - Made the Rigidbody component, as well as all collider components. Made them able to be added to objects to apply physics to them.
  - Made the Dynamic Physics System in order to run all the physics. The physics system and components are integrated so that the components are all added when loading a level, and the physics system starts if there is an object with a rigidbody in the level.
  - Made all joint components, also integrated so that they can be made when loading from a JSON file.
  - Made the raycasting classes, the ability to make a permanent raycast to check every frame, or a temporary one which is then destroyed. Worked with Jack to integrate into the lua scripting.
- Changed light and camera to be components.
  - Changed cameras so that they work in the components system. Made camera components to replace the camera types there were before.
    - Added the ability for cameras in the level to be able to see certain layers and not others.
  - Added light component so instead of one light being in the scene, lights can be on objects.
    - Added directional light. A light component has an ambient colour, point light diffuse and specular colours, and directional light diffuse and specular colours.
    - Directional light is controlled by the object's rotation. Point light is controlled by the objects position.
- Integrated ASSIMP model loading so objects can have a mesh component. Added to JSON loading.
- Added different texture types to the texture component.
  - A texture component can give an object 4 types of textures. An object can have a diffuse texture for colour, a specular map, a normal map, and a parallax map.
  - Changed the shader used by objects to use these textures.
- Added multithreading for various things.
  - Loading models can use multiple threads.
  - Physics updates can be run on a separate thread.
  - Updating objects can be done on a separate thread. This is also done on the physics update thread when it is not doing physics.
  - The computer’s hardware concurrency value is checked, and separate threads are only used if the computer can handle it without slowing down by having to switch between threads. Objects with lua scripts must be updated in order on the main thread.
- Helped others understand and use the engine.
  - Helped Adam implement shadow mapping and adapt layer updating to enable post-processing with multiple render passes.
  - Helped Adam change the texture slot assignment so that it can be used for frame buffer texture attachments.
  - Helped to understand the game object/component structure so they can add their things.
  - Helped Jack with accessing and changing variables of different components from a lua script component.
- Built the main structure for the functionality of the level editor. Integrated the profiler into the level editor.

(name):
Adam Hubble

(Contribution)

- Implemented post-processing renderer, separate from simple and basic renderers and is abstracted out into a series of classes.

- Refactored texture base class functionality to assign texture slots dynamically for the use of frame buffer colour and depth attachments, this was led with Alfie's support.
- Implemented frame buffer objects in abstract classes, which provide post-processing effects to the game in the sandbox and level editor application windows, via shader source code. Frame buffer objects support colour and depth attachments, multiple colour attachment support has been implemented but not used.
- Implemented a range of frame buffer effects as shader source code, which statically and dynamically affect the viewport of the game; effects can be toggled using key pressed events, some effects demonstrate various kernel effects which can be toggled independently of other functionality. Advanced post-processing effects consider night vision, varying depth-of-field (DOF) and the combination of shake, confuse and chaos effects interchangeably.
- Implemented wireframe mode, independent of shader source code by using OpenGL functionality.
- Implemented shadow mapping with the help from Alfie, the engine makes use of multiple pass rendering to achieve this. Shadow mapping also incorporates a bias through shader source code to correctly light the game scene.
- Implemented multi-sampled anti-aliasing (MSAA), where its activeness and level of detail can be toggled independently of other functionality. MSAA levels adhere to the graphical capabilities of the running system also.
- Built and integrated the performance profiler tool in addition to the level editor tool.
- As project manager, all the management documentation was created and managed by me, as well as the arrangements directed for group meetings.

(name):
Jack Moorin

(Contribution)
- Created the Lua Script component and developed all the below functionality:
    - An attached Lua Script component contains a Start and Update function by default. The start function is run once when the object it's attached to is created, either at the start of the game or when the object is created from a prefab. The update function is run once every subsequent frame.
    - The attached object's transform can be gotten and set from within a script. Users can get and set the objects position, rotation and scale, either with the x, y and z elements directly or by using a LuaVector. There are also bespoke functions for translating, rotating, incrementing the scale and scaling by a factor.
    - Defined a LuaVector class that is used as an intermediary between the engine user's Lua scripts and the engines C++ functions. Users can set the x, y and z elements of a LuaVector, get its magnitude, normalize it or scale it by a factor. Users can also add LuaVectors together, subtract them, multiply them and get the dot product or angle between two of them.
    - Defined a LuaColour class that is used similarly to a LuaVector except for colours. Users can set and create LuaColours from r, g, b values or a hexadecimal string.
    - Users can access the timestep between frames.
    - Users can set the model of the attached object, either by passing in the file path of a created model or a filepath to a given default model. These are a cube, a capsule, a cylinder, a quad and a sphere.
    - Users can use the engine's logger functionality to output to the console window.
    - Users can check for input from keyboard keys or the mouse.
    - Users can alter the attached objects material. They can get and set it's diffuse, specular and shininess values.
    - Users can access the height scale of the attached objects texture component, if it has one.
    - Added various functionality regarding the attached object's rigid body component, if it has one. Users can apply forces either directly to the body or to a specific point, apply torque and get and set its velocity both angular and linear.
    - If the attached object has a light component, users can set its ambient colour, point diffuse, point specular, directional diffuse, directional specular and attenuation.
    - In conjunction with Alfie's raycasting, users can directly perform or register a raycast with the physics system, either from the object or a specific point. If a ray is registered with the physics system, users can alter the ray's start point, direction and length, disable and enable the raycast being updated and get back info about a collided object in a created RaycastHit object.
    - Defined a RaycastHit object that contains whether the associated raycast was successful, the distance to the hit object and its name and tag.
    - Created a fully working prefab system. Users can create game objects from a prefab object in a Lua script. They can either be instantiated at the prefabs set position or at another position passed into the InstantiateAt function.
    - Created a prefab object that holds the information needed to create an object at run time. A blank prefab can be created, or one can be created with either a cube, capsule, cylinder, quad or sphere

model. The information within the prefab object that can be set are the objects name, tag, model, position, rotation, scale, material diffuse, material specular, material shininess and the filepath to the texture that will be used in a created texture component. Users can add dynamic, static or kinematic rigid bodies and box, sphere, capsule or convex mesh colliders and set which other colliders it may collide with. Users can also set the filepath to a Lua script file to add a Lua script component to the created object.

- Wrote and maintained a detailed list of the Lua Script component's functionality throughout development. Help page can be found by clicking on the '?' button in the drop-down menu for Lua Script components in the level editor. How to make use of all the above functionality is explained to users and listed in helpful tables.
- Added the 'Open' button, also on the Lua Script component drop-down menu, that will open the currently selected Lua Script file when clicked.

(name):
Joesph Hill

(Contribution)

- Not applicable.

Describe the functionality of your tool:

- A level editor.
  - When the user opens it, they can start a new project, save what they've done and reload it using JSON files. A light and a camera are in the level by default.
  - 3D objects can be added from a list of default models, or the user can add a model to the projects model folder for it to be used in the editor. Cameras and lights can't be added.
  - All objects have a transform component by default, the position, rotation and scale can be changed.
  - The camera has a camera component and all the important things (which layers the camera can see, FOV etc.) can be changed. The light has a light component, with point light and directional light colours.
  - Other objects have material, texture and mesh (model) components. The model can be changed, the diffuse and specular colour can be changed, as well as the shininess. Diffuse, specular, normal map and parallax map textures can be added to all objects (they have to be added to the projects texture folder). If a parallax map is added, the height variable can be changed. Luascript components can be added to objects, an existing script can be chosen, or a new one can be made, the script can be opened from the component section. Collider and rigidbody components can also be added, and all necessary variables can be edited. When the collider component dropdown is open on the selected object, the collider is drawn.
  - Non-necessary components can be deleted from objects, and objects (except the light and camera) can be deleted.
  - All editing is done through menus using ImGui, the user can change the layout of the menus using ImGui docking.
  - The current scene is rendered in the scene view tab, and the user can move the camera around in order to see what they want. The view from the camera in the game can be seen in the game view.
  - Objects have a name, a tag and a layer that they are in, all of these can be changed.
  - Joints can be added to the level, given a position and the names of the two objects they connect. The objects and the joint must be in the same layer.
  - Text can be added and given all relevant information (fonts, sizes, actual text, colour). Text can't be seen when not running the game.
  - The game can be run from the editor. When the run button is pressed, the project is saved, and new JSON files are made which are loaded using the engine to make the game run. The running game is viewed from the game view tab and the game can be played here. When the game is running, nothing in the editor can be changed. When the stop button is pressed, the game stops running, and the editor is back to how it should be.
  - Object tags can be made and deleted. New 2D and 3D layers can be made for objects, they can also be re-named and deleted.
  - New levels can be made and changed between.
- A performance profiler
  - Primarily displays the number of frames passing per-second, which can be alternated as an average or live calculation, by using the ImGui slider feature to govern the number of frames considered in the averaging.
  - The number of frames passing per-second is illustrated by the implementation of a line graph, that updates on a timed basis for readability purposes. Textual overlays also exist and share a similar update basis with the line graph, for presenting data regarding FPS, linguistically.

○ Horizontal bar chart widget is featured below the FPS graph and slider widgets, for identifying the number of frames dropped over the duration of runtime; as a fraction of the number of frames created, over the duration of runtime. Like the line graph widget, textual overlays are provided for linguistic interpretation of the data. ○ A nested drop-down interface is provided for identifying the execution times of a range of engine processes, at the start and throughout runtime.
Describe the functionality of your game:
The deliverable state of the game engine does not feature a game, as overridden in previous developments of the module.
State what software development methodology you have used and how is has been applied:
During the 'Team Funk' game engine's development we adhered to the agile development methodology, specifically SCRUM. We chose to use this methodology because due to its limited development time, making small but consistent increments to its overall functionality meant that whenever we reached a deadline, we would always have a recent working build of the game we could present. Agile development also fits well with the fact that during the limited development time we had for this project we were also doing work for our other modules as well, as its easier to pick up and get back into smaller developmental increments than larger ones.
What problems have you encountered and how have you overcome them:
The biggest problem faced throughout the engine's development was that one of our group members, Joe Hill, left during it. Unfortunately, this meant that we couldn't include his planned component, audio, however, we didn't need it for the engine to work fortunately.

Appendices

Please ensure you include the following:

- Project backlog (can be in the form of a Kanban board)
- Sprint organisation
- Burndown chart
- Groups meeting minutes (including date, time, attendance, summary of discussion and actions)
- Time records

For the project management documentation, see: https://github.com/IMAT3905/cw-2019-2020-team-funk/tree/master/ProjectManagement, within the "cw-2019-2020-team-funk/ProjectManagement/" directory. The file named 'GroupManagement.xlsx' contains all of the above.