D7049E - Assignment report

Author
`Alex Peschel`
Raphaël Dupont
Josef Vilestad
Oliver Olofsson

Virtual interactive environments

May 25, 2022

Contents

1 Introduction

This simple Game Engine was created as a school project at Luleå University of Technology in Sweden. It is made as a project for the course "D7049E, Virtual interactive environments". In the end the Game Engine is the goal, however to ease up and specialize the game engine it will be made to create a battle simulator. Here with focus of multiple entities going against multiple other entities with complex prioritization trees.

2 Executive summary recap

Following the start of the course we as a group needed to elaborate what programming language and game idea to strive for. Following some deductions within the group we decided on c++ as the coding language. A large part of the decision why we decided on c++ were because of the previous lack of knowledge within the language. Alex and Oliver had no previous experience in the language while Josef and Raphaël had some previous experience. As a result the project felt as a good way to learn the new language with actual implementations. As for the game idea, the basis came from Josef. Josef pitched Isometric and top down games. From that as group we talked and decided on a top down battle simulator similar to age of empire without the buildings.

The initial system design was mostly dependant on two classes, Game and Application. The purpose of game is to instantiate the application class in GameEngine and this structure has remained to the end product. A logging system and a simple input management system were the only features implemented to have a test bed for implementing different libraries.

During the first revisions of the system structure, some parts connected to the initialization were deemed unnecessary and therefor removed. Parts that were added at this stage was a window, implemented with the SDL library. SDL was chosen as it would work well with the planed addition of rendering libraries like filament and BGFX. It was also deemed better to switch audio library from SoLoud to SDL mixer as the latter could be implemented easier.

The next set of changes made to the system was the simplification of the input events. Instead of having multiple files which kept track of different inputs, all of them were gathered into one file. This input manager now handled both the keyboard buttons and mouse buttons as well as the mouse position all in one. This made easier to implement and used with the SDL window, however, should future work expand the size of the system it might be better to split the input manager into separate parts again.

Additional changes were the move from filament to BGFX that was made because of the continues problems that kept occurring with errors and lack of documentation. BGFX on the other hand had more documentation and tutorials to work which made the change fairly quick. A MathHelper file was added to give the possibility to create and work with vectors throughout the system. The vectors was extremely imported for when working with the 3D space, with placements and movements.

3 System's limitations and the possibilities for improvements.

The current instance of the game engine has a large amount of possibilities for the future. However, at the moment it also has a couple of limitations where the system has had problem to achieve the indented outcome. Non the less, it could also be due to several functionalities not included into the engine just yet.

3.1 Limitations

The current largest limitations within the system is the following. First of all we can't create a entity exactly where the mouse is if we move the camera from the original position. Following that we can't move the entity in correlation to their rotational direction. What is meant here is, if an entity looks left it does not change it's own x or z direction it acts as if it looked to the front still. As a third limitation, the coalition is not optimal at the moment. It takes a lot of computing power and following that it lags when there is a large amount of units present. They also don't really move away from one another but can still go through each other when colliding. We can't save multiple sound files in one unit, meaning they are limited to one sound. The navmesh movement has no correlation to physics implemented.

3.2 Possibilities

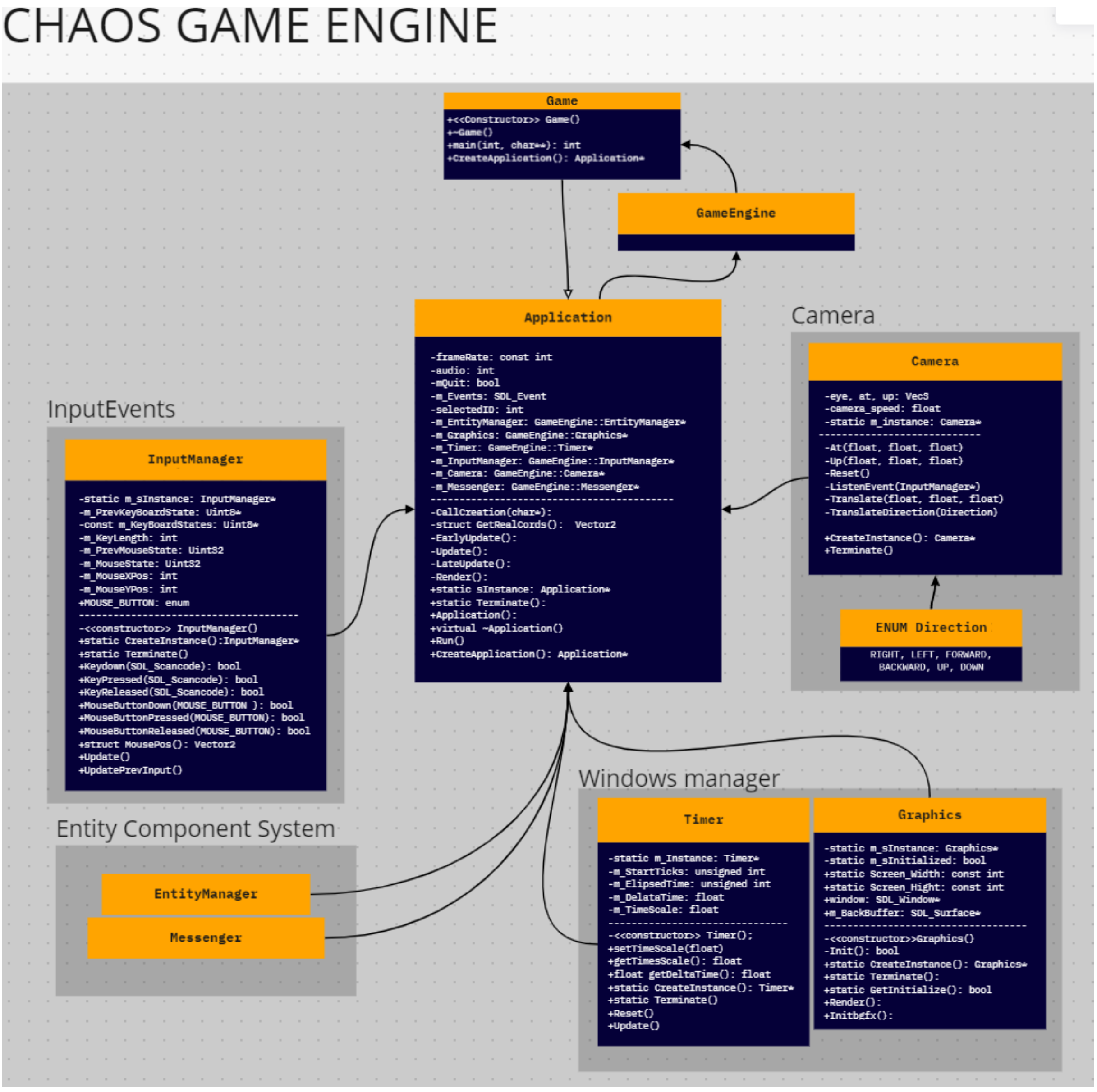
Though with all limitations a hand full of possibilities are here for the future. Everything in the Limitations section can be solved and added to improve the engine as a whole. With the current entity system there is a great possibility to add more components. Ad a id generation pool instead of a global value deciding on the id one of those. After all we have already thought of some components such as animations, light and a more developed coalition component. We could also include more libraries into the system such as physics. We have created a small version of it but for further development either include a library or create it fully from scratch. However, one of the largest improvements which can occur which in turn will help include other functionaries is the GUI. With the GUI we planed on including the possibility to add components dynamical to an entity and storing it in a json file. While also adding entity's into the system without pressing specific button for each entity. A large potion of it will surround json files but also the creation and addition of new and old entity's with new and old components.

There's many possibilities for improvement, but these are the mayor once that has been discussed at least once before.

4 System architecture description and overview of the implementation.

The following subsections within section 4 included images and small descriptions of the chaos engine different components.

In figure 1 we have a uml diagram showing the current game engine. However, I will note that we have more code within Entity Component system, this figure is only the a representation of the connection to make it easier to understand. For further read and understanding go to 4.1. We have kept the system in somewhat the same structure throughout the course and believe we are happy with the current structure and the current functionalities. As a note, GameEngine is mainly a file where we have several inclusions(#include) so that Game has everything it needs.

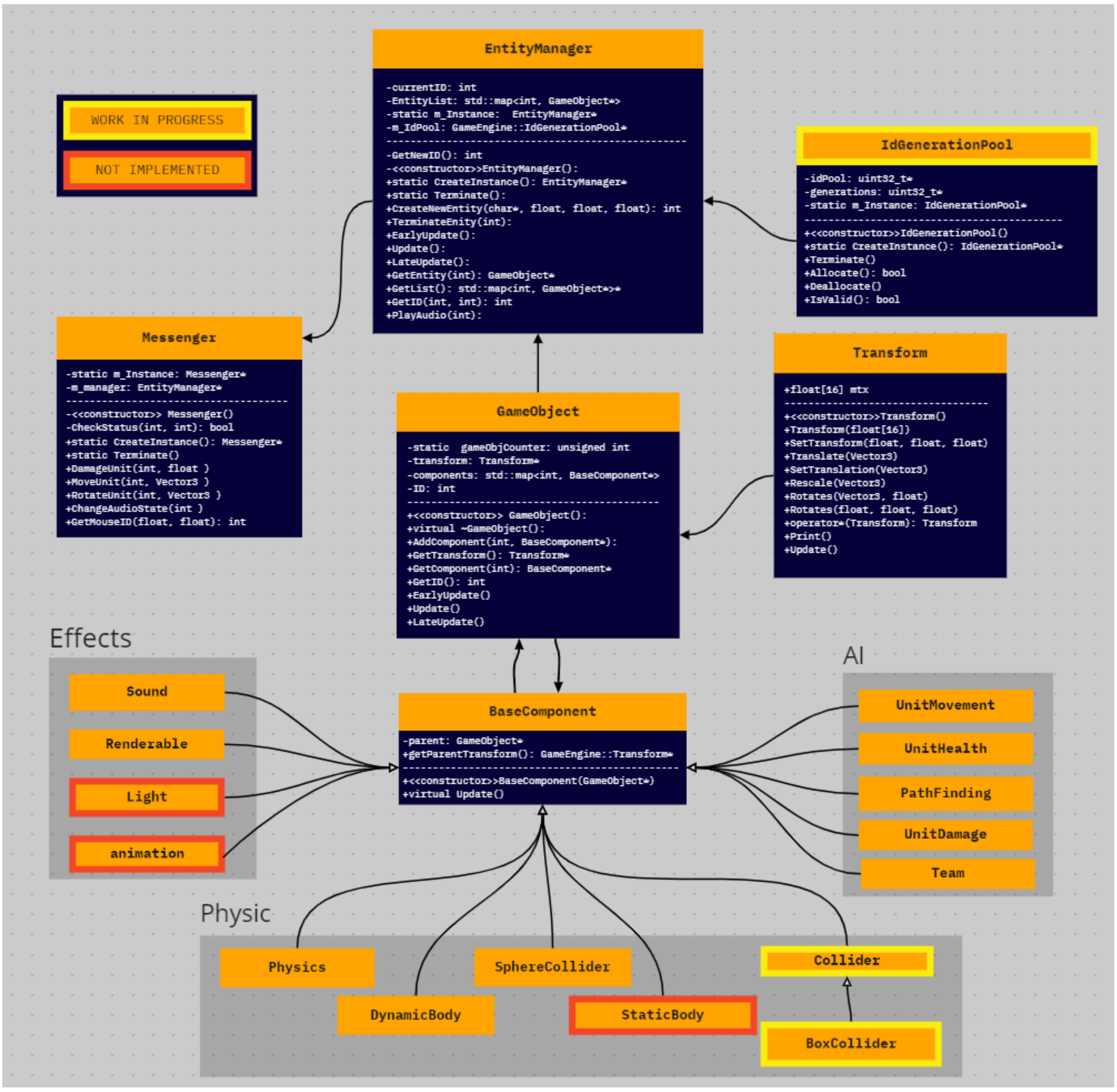


CHAOS GAME ENGINE
Game
+<<Constructor>> Game()
+~Game()
+main(int, char**): int
+CreateApplication(): Application*
GameEngine
Application
-frameRate: const int
-audio: int
-mQuit: bool
-m_Events: SDL_Event
-selectedID: int
-m_EntityManager: GameEngine::EntityManager*
-m_Graphics: GameEngine::Graphics*
-m_Timer: GameEngine::Timer*
-m_InputManager: GameEngine::InputManager*
-m_Camera: GameEngine::Camera*
-m_Messenger: GameEngine::Messenger*
-CallCreation(char*):
-struct GetRealCords(): Vector2
-EarlyUpdate():
-Update():
-LateUpdate():
-Render():
+static sInstance: Application*
+static Terminate():
+Application():
+virtual ~Application()
+Run()
+CreateApplication(): Application*
Camera
Camera
-eye, at, up: Vec3
-camera_speed: float
-static m_instance: Camera*
-At(float, float, float)
-Up(float, float, float)
-Reset()
-ListenEvent(InputManager*)
-Translate(float, float, float)
-TranslateDirection(Direction)
+CreateInstance(): Camera*
+Terminate()
ENUM Direction
RIGHT, LEFT, FORWARD, BACKWARD, UP, DOWN
InputEvents
InputManager
-static m_sInstance: InputManager*
-m_PrevKeyBoardState: Uint8*
-const m_KeyBoardStates: Uint8*
-m_KeyLength: int
-m_PrevMouseState: Uint32
-m_MouseState: Uint32
-m_MouseXPos: int
-m_MouseYPos: int
+MOUSE_BUTTON: enum
-<<constructor>> InputManager()
+static CreateInstance():InputManager*
+static Terminate()
+Keydown(SDL_Scancode): bool
+KeyPressed(SDL_Scancode): bool
+KeyReleased(SDL_Scancode): bool
+MouseButtonDown(MOUSE_BUTTON): bool
+MouseButtonPressed(MOUSE_BUTTON): bool
+MouseButtonReleased(MOUSE_BUTTON): bool
+struct MousePos(): Vector2
+Update()
+UpdatePrevInput()
Entity Component System
EntityManager
Messenger
Windows manager
Timer
-static m_Instance: Timer*
-m_StartTicks: unsigned int
-m_ElipsedTime: unsigned int
-m_DelataTime: float
-m_TimeScale: float
-<<constructor>> Timer();
+setTimeScale(float)
+getTimesScale(): float
+float getDeltaTime(): float
+static CreateInstance(): Timer*
+static Terminate()
+Reset()
+Update()
Graphics
-static m_sInstance: Graphics*
-static m_sInitialized: bool
+static Screen_Width: const int
+static Screen_Hight: const int
+window: SDL_Window*
+m_BackBuffer: SDL_Surface*
-<<constructor>>Graphics()
-Init(): bool
+static CreateInstance(): Graphics*
+static Terminate():
+static GetInitialize(): bool
+Render():
+Initbgfx():

Figure 1: Chaos UML diagram

4.1 Entity System

As you can see figure 2 Chaos engine is an entity component system. It means than you can create entities (GameObject) and define there behavior with components. Chaos also provides tools to create and manage an application, a window and input events. At first our entity system design was supposed to have much more components but regarding the time left at the end of the project we had to lower our expectation and implement only the core functionalities of our game engine.

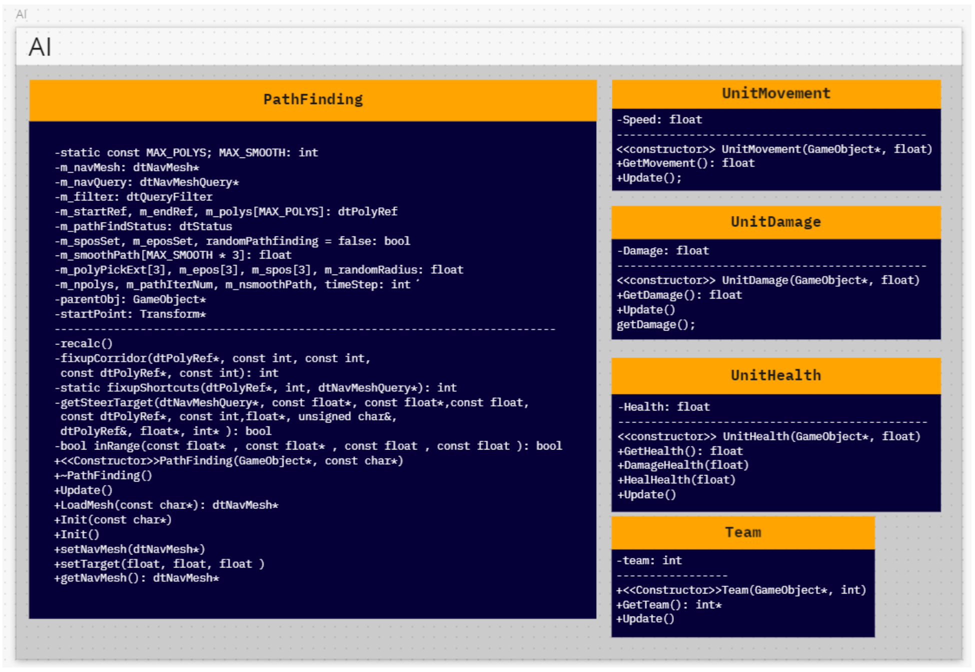


WORK IN PROGRESS
NOT IMPLEMENTED
EntityManager
-currentID: int
-EntityList: std::map<int, GameObject*>
-static m_Instance: EntityManager*
-m_IdPool: GameEngine::IdGenerationPool*
-GetNewID(): int
-<<constructor>>EntityManager():
+static CreateInstance(): EntityManager*
+static Terminate():
+CreateNewEntity(char*, float, float, float): int
+TerminateEnity(int):
+EarlyUpdate():
+Update():
+LateUpdate():
+GetEntity(int): GameObject*
+GetList(): std::map<int, GameObject*>*
+GetID(int, int): int
+PlayAudio(int):
IdGenerationPool
-idPool: uint32_t*
-generations: uint32_t*
-static m_Instance: IdGenerationPool*
+<<constructor>>IdGenerationPool()
+static CreateInstance(): IdGenerationPool*
+Terminate()
+Allocate(): bool
+Deallocate()
+IsValid(): bool
Messenger
-static m_Instance: Messenger*
-m_manager: EntityManager*
-<<constructor>> Messenger()
-CheckStatus(int, int): bool
+static CreateInstance(): Messenger*
+static Terminate()
+DamageUnit(int, float)
+MoveUnit(int, Vector3)
+RotateUnit(int, Vector3)
+ChangeAudioState(int)
+GetMouseID(float, float): int
Transform
+float[16] mtx
+<<constructor>>Transform()
+Transform(float[16])
+SetTransform(float, float, float)
+Translate(Vector3)
+SetTranslation(Vector3)
+Rescale(Vector3)
+Rotates(Vector3, float)
+Rotates(float, float, float)
+operator*(Transform): Transform
+Print()
+Update()
GameObject
-static gameObjCounter: unsigned int
-transform: Transform*
-components: std::map<int, BaseComponent*>
-ID: int
+<<constructor>> GameObject():
+virtual ~GameObject():
+AddComponent(int, BaseComponent*):
+GetTransform(): Transform*
+GetComponent(int): BaseComponent*
+GetID(): int
+EarlyUpdate()
+Update()
+LateUpdate()
BaseComponent
-parent: GameObject*
+getParentTransform(): GameEngine::Transform*
+<<constructor>>BaseComponent(GameObject*)
+virtual Update()
Effects
Sound
Renderable
Light
animation
AI
UnitMovement
UnitHealth
PathFinding
UnitDamage
Team
Physic
Physics
SphereCollider
Collider
DynamicBody
StaticBody
BoxCollider

Figure 2: UML diagram of the Entity System

4.1.1 AI

In figure 3 we have the currently accessible and used classes and methods. The AI folder includes classes which has something to do with how the unit acts or its stats. In our case we have the PathFinding class which helps calculate how units walk from point a to b. While the rest are mainly containing information regarding the different areas such as health, damage and team.



```
PathFinding

-static const MAX_POLYS; MAX_SMOOTH: int
-m_navMesh: dtNavMesh*
-m_navQuery: dtNavMeshQuery*
-m_filter: dtQueryFilter
-m_startRef, m_endRef, m_polys[MAX_POLYS]: dtPolyRef
-m_pathFindStatus: dtStatus
-m_sposSet, m_eposSet, randomPathfinding = false: bool
-m_smoothPath[MAX_SMOOTH * 3]: float
-m_polyPickExt[3], m_epos[3], m_spos[3], m_randomRadius: float
-m_npolys, m_pathIterNum, m_nsmoothPath, timeStep: int ´
-parentObj: GameObject*
-startPoint: Transform*
--------------------------------------------------------------------------
-recalc()
-fixupCorridor(dtPolyRef*, const int, const int,
 const dtPolyRef*, const int): int
-static fixupShortcuts(dtPolyRef*, int, dtNavMeshQuery*): int
-getSteerTarget(dtNavMeshQuery*, const float*, const float*,const float,
 const dtPolyRef*, const int,float*, unsigned char&,
 dtPolyRef&, float*, int* ): bool
-bool inRange(const float* , const float* , const float , const float ): bool
+<<Constructor>>PathFinding(GameObject*, const char*)
+~PathFinding()
+Update()
+LoadMesh(const char*): dtNavMesh*
+Init(const char*)
+Init()
+setNavMesh(dtNavMesh*)
+setTarget(float, float, float )
+getNavMesh(): dtNavMesh*
```

```
UnitMovement

-Speed: float
----------------------------------------------
<<constructor>> UnitMovement(GameObject*, float)
+GetMovement(): float
+Update();
```

```
UnitDamage

-Damage: float
----------------------------------------------
<<constructor>> UnitDamage(GameObject*, float)
+GetDamage(): float
+Update()
getDamage();
```

```
UnitHealth

-Health: float
----------------------------------------------
<<constructor>> UnitHealth(GameObject*, float)
+GetHealth(): float
+DamageHealth(float)
+HealHealth(float)
+Update()
```

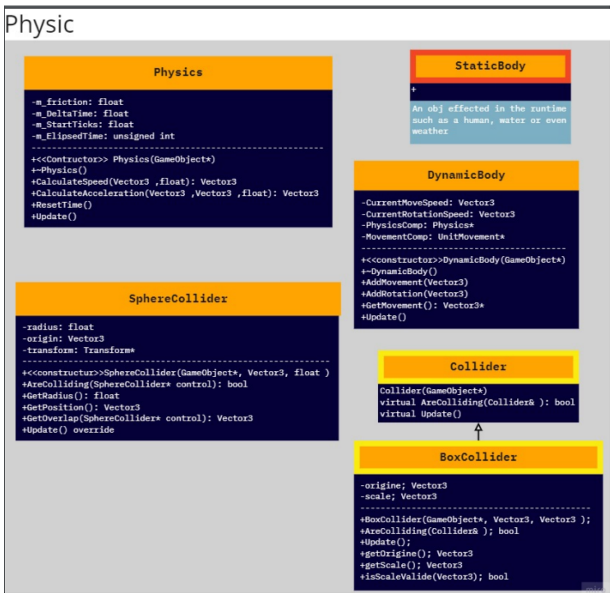
```
Team

-team: int
----------------
+<<Constructor>>Team(GameObject*, int)
+GetTeam(): int*
+Update()
```

Figure 3: UML diagram of the AI folder

4.1.2 Physic

The physic folder contains the parts which would take in regards any physics systems. This could either be coalitions or accurate movement in correlation to friction. However, as can be seen in figure 4 there are several components in this area which is either not included or have yet been created. This is mainly due to the fact that physic was mainly focused on the last couple of days just before the presentation.



```
Physics

-m_friction: float
-m_DeltaTime: float
-m_StartTicks: float
-m_ElipsedTime: unsigned int
-------------------------------------------------------
+<<Contructor>> Physics(GameObject*)
+~Physics()
+CalculateSpeed(Vector3 ,float): Vector3
+CalculateAcceleration(Vector3 ,Vector3 ,float): Vector3
+ResetTime()
+Update()
```

```
StaticBody

+
An obj effected in the runtime
such as a human, water or even
weather
```

```
DynamicBody

-CurrentMoveSpeed: Vector3
-CurrentRotationSpeed: Vector3
-PhysicsComp: Physics*
-MovementComp: UnitMovement*
----------------------------------------
+<<constructor>>DynamicBody(GameObject*)
+~DynamicBody()
+AddMovement(Vector3)
+AddRotation(Vector3)
+GetMovement(): Vector3*
+Update()
```

```
SphereCollider

-radius: float
-origin: Vector3
-transform: Transform*
-------------------------------------------------------------
+<<constructur>>SphereCollider(GameObject*, Vector3, float )
+AreColliding(SphereCollider* control): bool
+GetRadius(): float
+GetPosition(): Vector3
+GetOverlap(SphereCollider* control): Vector3
+Update() override
```

```
Collider

Collider(GameObject*)
virtual AreColliding(Collider& ): bool
virtual Update()
```

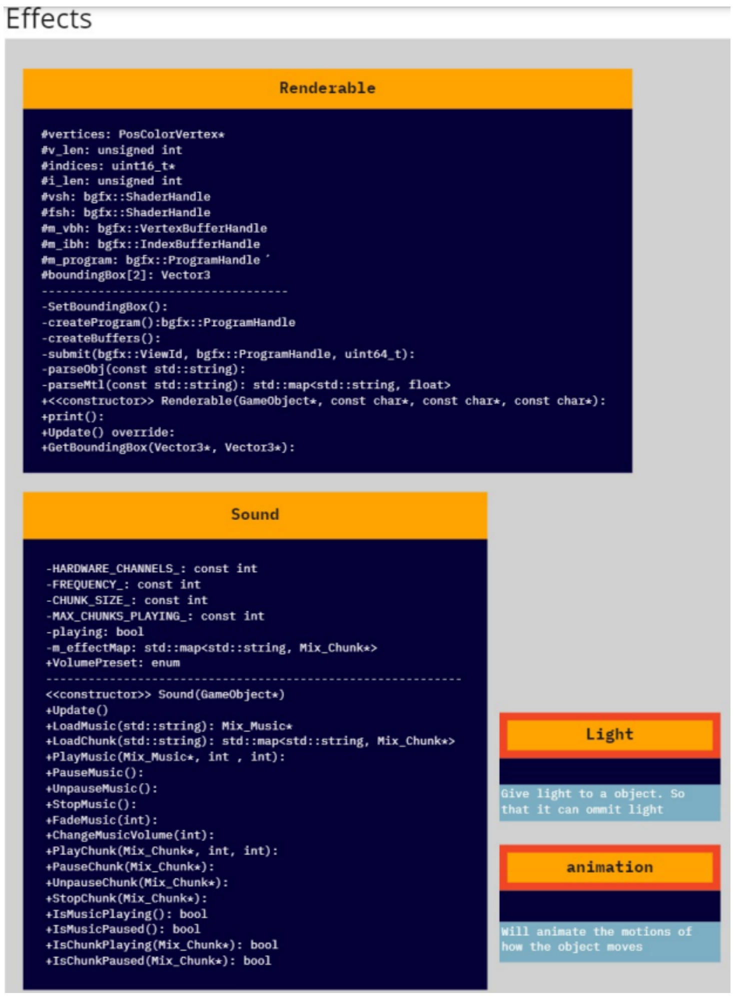
```
BoxCollider

-origine; Vector3
-scale; Vector3
---------------------------------------------
+BoxCollider(GameObject*, Vector3, Vector3 );
+AreColliding(Collider& ); bool
+Update();
+getOrigine(); Vector3
+getScale(); Vector3
+isScaleValide(Vector3); bool
```

Figure 4: UML diagram of the Physics folder

4.1.3 Effects

The effects components include the following parts represented in figure 5. These components are mainly effects which gives the player better feel for the game. Sound to create a background noise or sound when you interact with something. However, one of the largest part is how we as the user see things. In effects we have the renderable, which give the possibility to see a unit, while also animation and light. Due to time constraints animation and light was not able to be included at this time but if they could have been, they would have been included in effects.



```
#vertices: PosColorVertex*
#v_len: unsigned int
#indices: uint16_t*
#i_len: unsigned int
#vsh: bgfx::ShaderHandle
#fsh: bgfx::ShaderHandle
#m_vbh: bgfx::VertexBufferHandle
#m_ibh: bgfx::IndexBufferHandle
#m_program: bgfx::ProgramHandle '
#boundingBox[2]: Vector3
-----------------------------------
-SetBoundingBox():
-createProgram():bgfx::ProgramHandle
-createBuffers():
-submit(bgfx::ViewId, bgfx::ProgramHandle, uint64_t):
-parseObj(const std::string):
-parseMtl(const std::string): std::map<std::string, float>
+<<constructor>> Renderable(GameObject*, const char*, const char*, const char*):
+print():
+Update() override:
+GetBoundingBox(Vector3*, Vector3*):
```

```
-HARDWARE_CHANNELS_: const int
-FREQUENCY_: const int
-CHUNK_SIZE_: const int
-MAX_CHUNKS_PLAYING_: const int
-playing: bool
-m_effectMap: std::map<std::string, Mix_Chunk*>
+VolumePreset: enum
-----------------------------------------------------------
<<constructor>> Sound(GameObject*)
+Update()
+LoadMusic(std::string): Mix_Music*
+LoadChunk(std::string): std::map<std::string, Mix_Chunk*>
+PlayMusic(Mix_Music*, int , int):
+PauseMusic():
+UnpauseMusic():
+StopMusic():
+FadeMusic(int):
+ChangeMusicVolume(int):
+PlayChunk(Mix_Chunk*, int, int):
+PauseChunk(Mix_Chunk*):
+UnpauseChunk(Mix_Chunk*):
+StopChunk(Mix_Chunk*):
+IsMusicPlaying(): bool
+IsMusicPaused(): bool
+IsChunkPlaying(Mix_Chunk*): bool
+IsChunkPaused(Mix_Chunk*): bool
```

Give light to a object. So that it can ommit light

Will animate the motions of how the object moves

Figure 5: UML diagram of the Effects folder

4.2 Optimisation

Following we have several optimisations which has been done within the code. Some on a smaller and less noticeable impact while some had a large impact which could be greatly noticeable when running the code.

One of the large optimisations which was noticeable within the code was the implementation of data oriented storage. What I mean hear is the implementation of a json file which kept the information regarding how a specific unit type needs in the form of components. This has given the possibility for future implementation where code it self add more data into the json file instead of the user doing it by hand. As well as resulting in less code within the specific entity creation. Mainly in the way that each entity does not need specific code for each instance, but rather share a common dynamic one.

Another optimisation that allowed us to load thousand of units in our game was to add a cache into our *Renderable* class. Before this optimisation we used to load the 3D models from the .obj and .mtl files each time we were using it in our environment. Now, we're using a map as a cache to store the meshes that are already loaded in our scene which key is the name of the path of the 3D model and which value is the corresponding *Renderable*. To create a *Renderable* we load it from the cache if it's already in our scene or else we open the .obj and .mtl files and add it to the cache.

A change was made to implement the navmesh in the world map object instead of each entity containing its own copy. This does add some overhead when creating units but it resolves a lot of issues.

5 Link to code

5.1 Git

The Chaos game engine is available in this GitHub Repository. To setup Chaos you can follow the readme instructions of refer to the section 5.2

5.2 Installation

5.2.1 Prerequisites

Chaos was made to run on *Windows* and does not provide any version that can be run on *Linux* or *MacOS*.

5.2.2 Setup

Start of by cloning the git repository onto your computer. Then go into the chaos folder and open a terminal there. Here we want to download the libraries used in the project. So to download the sub modules with ease in the right place use the following command *"git submodule update –init –recursive"* within the main directory. Then clone SDL2, jsoncpp, bx, bgfx and bimg into the same folder, *"GameEngine/library"*. When SDL mixer has been downloaded, place its header and lib files in the corresponding directories inside SDL2. References to the repositories can be seen in the section 5.2.3.

You will need to run the Jsoncpp. Here you should run the cmake file within the folder or follow this link to build the jsoncpp library.

To build RecastNavigation run Cmake and place the output in the same recast folder, use the SDL library already downloaded from git submodules. Open up the project in visual studio and change the runtime library field for each library project to MTd in properties->C/C++->Code Generation . Now you can build the libraries.

However, due to how bgfx is built up we will need to fix some things within the folder. So open bgfx, then open a command prompt within the main directory of bgfx. Here run *"../bx/tools/bin/windows/genie –with-examples –with-tools vs2017"*.

You should now have a *"./Build"* folder, if not it may be hidden so put on the possibilities to see hidden files/folders in the view settings. Then you wanna go to *".build/projects/vs2017/bgfx"*.sln and start it up. When starting it for the first time you may get asked to update the project, do it. Then when it is update build the project. If you wanna make sure it works as intended run example 14 by setting it as the "startup project" you can also try to run shaderc if you want to compile new shaders later on (see section Compile shaders below). If it works you should have prepared the bgfx library correctly.(for the instructions from bgfx them self follow this link)

When the download has been done it is time to run *"GenerateProject.bat"* in the main directory. This will make the necessary connections and links within the solution.

Then Start up the project with the sln file and you should be good to go.

5.2.3 Used Projects

- https://github.com/gabime/spdlog.git
- https://www.libsdl.org/download-2.0.php (development library, SDL2-Devel-2 0.20-VC.zip)
- https://www.libsdl.org/projects/SDL_mixer/ (Development library, SDL2_mixer-devel-2.0.4-VC.zip)
- https://github.com/bkaradzic/bx.git
- https://github.com/bkaradzic/bimg.git
- https://github.com/bkaradzic/bgfx.git
- https://github.com/open-source-parsers/jsoncpp
- https://github.com/recastnavigation/recastnavigation

5.2.4 Compile shaders

This section explain how to create new shaders and to compile them using bfx library. If you only want to use default shaders you can skip this section.

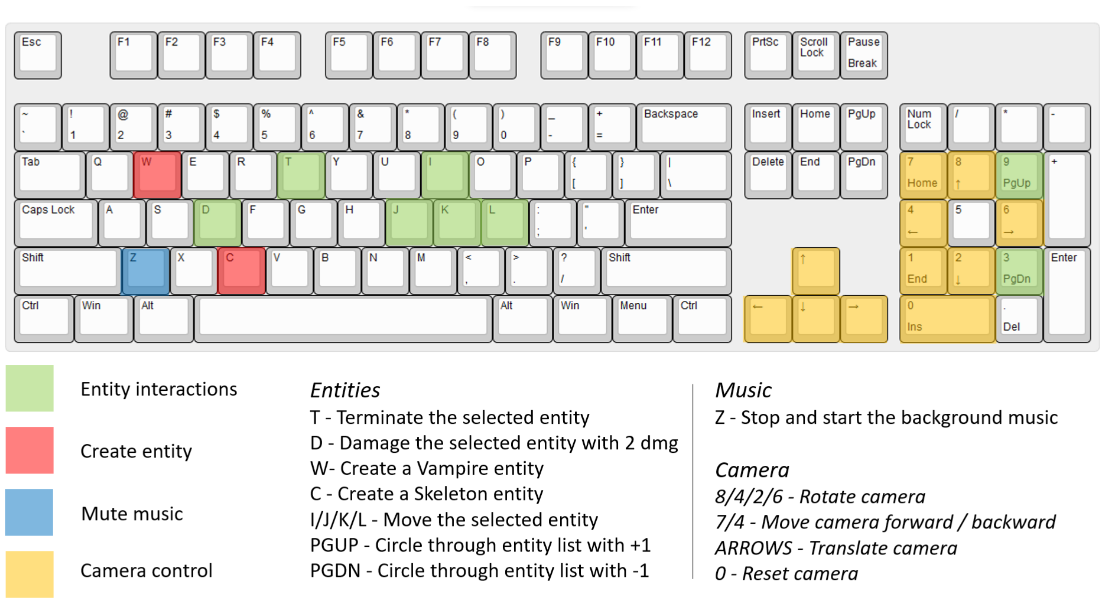
The existing shaders are located in */Game/Data/Shaders*. The fragment shader and vertex shader use in Chaos to render basic 3D model are shader from the bgfx repository shader folder. If you want to create your own shader you can find some example of .sc shader source files on this website. Then you have to use the tool shaderc to compile the shader source files into binary shader files that can be read by bgfx. Shaderc tool is located in *"GameEngine/library/bgfx/.build/win64_vs2017/bin/shadercDebug.exe"* and you can compile shaders as followed :

```
shadercDebug.exe -f intputShader.sc -o output.bin --platform windows --type shaderType
--varyingdef varying.def.sc
```

To find more information about shaderc please read the corresponding documentation

5.3 Usage

You can find figure 6 the list of the application shortcuts



🟩 Entity interactions

🟥 Create entity

🟦 Mute music

🟨 Camera control

Entities
T - Terminate the selected entity
D - Damage the selected entity with 2 dmg
W- Create a Vampire entity
C - Create a Skeleton entity
I/J/K/L - Move the selected entity
PGUP - Circle through entity list with +1
PGDN - Circle through entity list with -1

Music
Z - Stop and start the background music

Camera
8/4/2/6 - Rotate camera
7/4 - Move camera forward / backward
ARROWS - Translate camera
0 - Reset camera

Figure 6: List of application shortcuts

6 Authors

- Alex Peschel
- Raphaël Dupont
- Josef Vilestad
- Oliver Olofsson

7 References

7.1 GitHub

[1] https://github.com/Alepes-8/Chaos

7.2 YouTube

[2] https://www.youtube.com/channel/UCCKlrE0p4IZxqBpq98KFBmw

[3] https://www.youtube.com/channel/UCQ-W1KE9EYfdxhL6S4twUNw