FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

PROJECT PROGRAMMEREN 2021

GAME SLIME VOLLEY IN C++

prof. dr. ir. Filip De Turck

pgm@lists.ugent.be

2de Bachelor Ingenieurswetenschappen:
Computerwetenschappen en Elektrotechniek

Academiejaar 2020-2021

UNIVERSITEIT
GENT

INHOUDSOPGAVE

1 INTRODUCTIE

Hieronder wordt kort het doel van het project geschetst en wordt een inleiding gegeven over een aantal aspecten die typisch aan bod komen bij game programming. Het advies is om dit zeker door te nemen zodat er voldoende achtergrondkennis aanwezig is om met de opgave te starten.

1.1 Doel van het project

In dit project is het de bedoeling om een spel genaamd 'Slime Volley' te implementeren in C++. De uitwerking van het project dient te gebeuren in groepen van twee personen. Het doel is om zoveel mogelijk aspecten van softwareontwikkeling aan bod te laten komen. Hierbij denken we behalve het programmeren zelf aan het gebruik van de Visual Studio Code editor, de integratie van bestaande libraries en het efficiënt samenwerken aan een project door middel van Git. Het project is gequoteerd op 6 van de 20 punten voor het vak Programmeren.

1.2 Game programming

Het programmeren van games verschilt op een aantal vlakken van het programmeren van traditionele applicaties. Hieronder worden de belangrijkste zaken overlopen.

1.2.1 Game loop

De game loop is het belangrijkste onderdeel van het spel aangezien deze ervoor zorgt dat het spel vlot blijft lopen, onafhankelijk van het feit of de gebruiker al dan niet input genereert. Dit in tegenstelling tot meer traditionele softwareprogramma's zoals tekstverwerkers of browsers die enkel reageren wanneer de gebruiker een actie triggert, bijvoorbeeld het tekenen van een menu wanneer de gebruiker op een knop klikt. Een game loop in pseudocode zou er als volgt kunnen uitzien:

Algorithm 1 Een generiek voorbeeld van een game loop.

```
1: while user does not exit do
2:     Capture user input
3:     Run (simple) AI
4:     Move objects
5:     Resolve collisions
6:     Render graphics
```

Hierbij wordt achtereenvolgens gecontroleerd welke input er door de gebruiker gegeven is, wordt de logica van het spel en de artificiële intelligentie (AI) uitgevoerd, wordt de beweging van verschillende objecten uitgevoerd, worden eventuele botsingen afgehandeld en worden de objecten opnieuw getekend. De game loop kan uiteraard verder verfijnd worden naarmate de ontwikkeling van het spel verloopt, maar de meeste spellen zijn op dit basisprincipe gebaseerd.

Game loops kunnen anders geïmplementeerd worden afhankelijk van het platform waarvoor ze ontwikkeld zijn. Een spel voor game consoles kan bijvoorbeeld alle hardware resources naar eigen wens gebruiken, terwijl een spel voor Windows uitgevoerd moet worden binnen de beperkingen van de process scheduler. Verder is het zo dat de meeste spellen multithreaded zijn, zodat de berekening van de AI losgekoppeld kan worden van de generatie van bewegende beelden in het spel. Dit creëert een kleine overhead, maar zorgt ervoor dat het spel efficiënter uitgevoerd kan worden op hyper-threaded of multicore processoren.

1.2.2 Bewegende beelden

Het visuele aspect is bij games zeer belangrijk: er moet een vloeiend beeld weergegeven worden, wil men het spel er goed doen uitzien. De theorie rond hoeveel beelden per seconde het menselijk oog minimaal moet zien om iets als een vloeiende beweging waar te nemen, is veel ingewikkelder dan meestal wordt aangenomen. Er zijn een aantal vuistregels die in de meeste gevallen lijken te kloppen:

- Hoe groter het aantal frames per seconde (FPS), hoe vloeiender de beweging;
- Hoe kleiner het verschil tussen opeenvolgende frames, hoe vloeiender de beweging.

Het is gemakkelijk om de snelheid van de game loop ook in frames per seconde uit te drukken, zodat er een maat is om mee te vergelijken. Het aantal frames per seconde duidt dan eigenlijk aan hoeveel keer per seconde de toestand van het spel aangepast wordt. Het aantal gegenereerde FPS is enerzijds gelimiteerd door de hardware van de computer, maar anderzijds ook door hoe de berekeningen die in de game loop gebeuren. Wanneer een bepaalde actie bijvoorbeeld een groot aantal berekeningen vraagt, zal de iteratie van de game loop langer duren en zal het aantal zichtbare FPS (tijdelijk) dalen.

1.2.3 Collision detection

Bij games wordt dikwijls gesproken over "collision detection", het detecteren van een botsing tussen twee of meerdere objecten. In elke iteratie van de game loop moet er, vóór het hertekenen van het nieuwe frame, gecontroleerd worden of er al dan niet een botsing veroorzaakt werd door het uitvoeren van de bewegingen van alle verschillende objecten. Dit kan bijvoorbeeld het tegen elkaar plaatsen zijn van de twee objecten in plaats van "in elkaar" of een botsing zijn die de objecten de tegenovergestelde richting opstuurt. Eens deze botsing gedetecteerd en opgelost is, kan het frame hertekend worden. In dit project zal collision detection zich beperken tot de tweedimensionale variant.

1.3 Entity-component-system framework

Uit ervaring hebben we geleerd dat een klassieke objectgeoriënteerde aanpak niet altijd de beste resultaten oplevert bij het ontwerpen van een game. Abstracties die het ene moment logisch lijken, kunnen later een hindernis vormen bij het toevoegen van nieuwe functionaliteiten. Voorbeelden hiervan zijn:

- Tekenen naar het scherm gebeurt door elke klasse een `Render()` methode te laten implementeren, maar dit zorgt ervoor dat dit aspect van het spel verspreid wordt over de volledige codebase, wat het moeilijker onderhoudbaar en uitbreidbaar maakt;
- Bij een objectgeoriënteerde aanpak wordt generieke functionaliteit geïmplementeerd in klassen waarvan de objecten kunnen overerven. Een entiteit in het spel zou bijvoorbeeld kunnen overerven van de klasse `InputController` om te kunnen reageren op input van de gebruiker. Deze werkwijze is echter problematisch wanneer bijvoorbeeld het gedrag van deze entiteit at runtime moet veranderen in functie van de AI en niet in functie van de gebruiker.

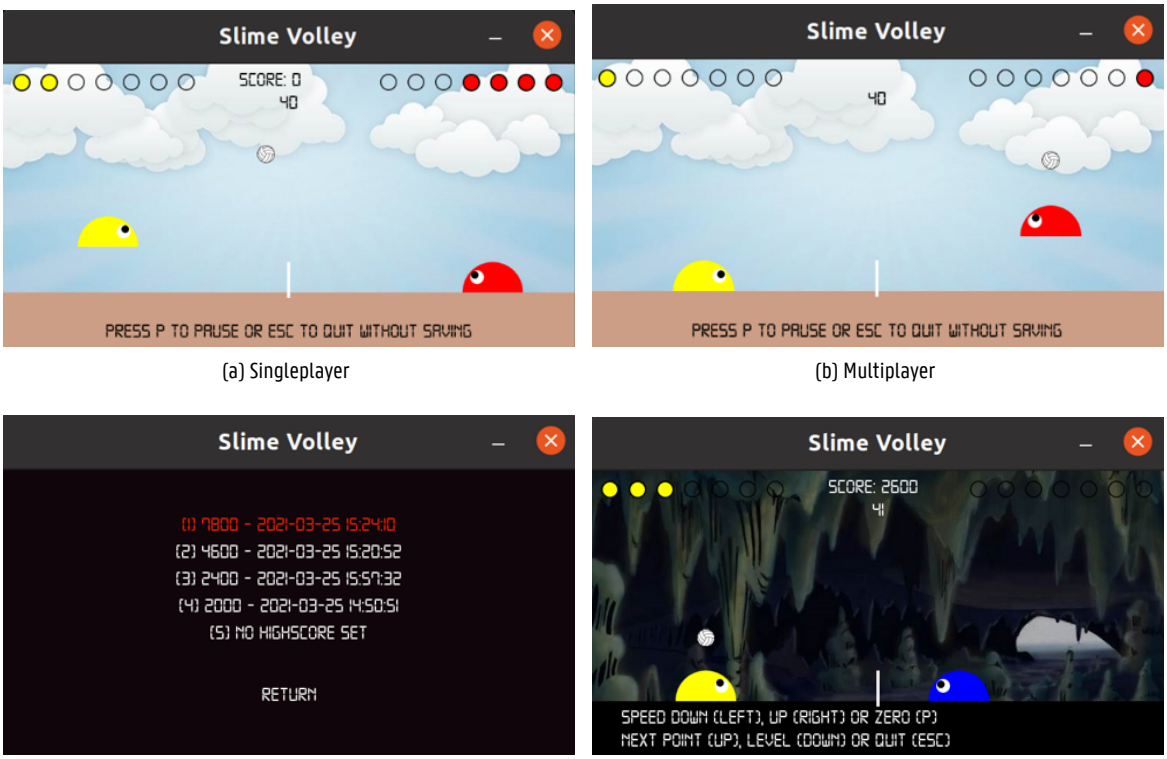
Bovenstaande problematiek wordt erkend in de spelindustrie en een van de voorgestelde oplossingen is het gebruik van een entity-component-system framework[1][2]. Bij deze data-driven aanpak worden de verschillende spelobjecten niet meer voorgesteld als afzonderlijke klassen, maar als generieke entiteiten die samengesteld worden aan de hand van een bepaald aantal componenten. Deze componenten zijn data containers die specifieke data bevatten, zoals bijvoorbeeld de positie en snelheid van een entiteit. De functionaliteit die per entiteit uitgevoerd wordt, wordt vervolgens bepaald door de verschillende systemen die elk inwerken op een set van entiteiten met specifieke componenten. Voorbeelden hiervan zijn een systeem dat bewegingsvectoren toepast en een systeem dat entiteiten op het scherm kan tekenen.

[1] http://www.richardlord.net/blog/why-use-an-entity-framework
[2] http://www.richardlord.net/blog/what-is-an-entity-framework

2 OPGAVE

2.1 Het spel

Het doel van het project is om een variant op het spel Slime Volleyball te implementeren. Zoals bij volleybal steeds het geval is, is het de bedoeling om punten te scoren door de bal te laten botsen op het veld van de tegenstander. De linkse speler begint steeds met op te slaan, en het spel is gedaan zodra een van de twee spelers zeven punten weet te scoren. Er wordt zowel een singleplayer als een multiplayer voorzien, zodat het spel zowel tegen de computer als tegen een echte tegenstander gespeeld kan worden. In de singleplayer zijn er drie mogelijke levels, waarbij de speler in levels 1, 2 en 3 respectievelijk 200, 400 en 600 punten krijgt voor elk gemaakt punt, en 100, 200 en 300 punten verliest voor elk punt tegen. Er wordt een lijst bijgehouden met daarin de vijf hoogst behaalde scores, en het is mogelijk om een van de scores aan te klikken en een volledige replay te zien te krijgen van het spel. Een aantal screenshots van onze geïmplementeerde versie zijn te zien in Figuur 1. Daarnaast zijn er ook een aantal video's beschikbaar die een deel van de voorbeeldoplossing laten zien[3] [4] [5] [6].



(a) Singleplayer

(b) Multiplayer

(c) Highscores

(d) Replay

Figuur 1: Vier verschillende use cases van het spel.

2.2 Allegro library

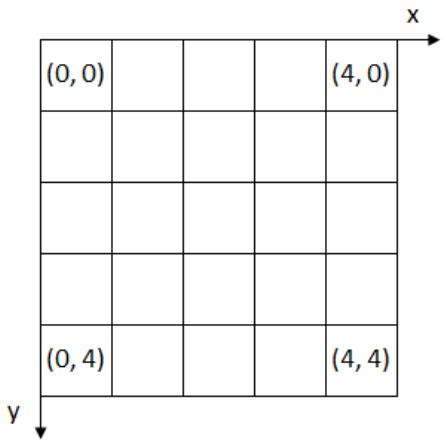
Om het visuele aspect van het spel af te handelen, zal er gebruik gemaakt worden van de Allegro library. Om deze library aan te roepen, wordt een AllegroLib klasse voorzien die de nodige funtionaliteiten abstraheert. Dit betreft alles in verband met initialisatie van Allegro, het opzetten van input, timing en events. Tracht zeker de code eens te bekijken, zodat het duidelijk is wat deze klasse precies aanbiedt. Meer informatie over de API en documentatie omtrent de Allegro library is terug te vinden via https://liballeg.org en https://www.allegro.cc.

[3] https://cloud.ilabt.imec.be/index.php/s/a64istpX6nzGeCN
[4] https://cloud.ilabt.imec.be/index.php/s/P9ZADz6idaYZHzP
[5] https://cloud.ilabt.imec.be/index.php/s/C7gfqZq4eASBckp
[6] https://cloud.ilabt.imec.be/index.php/s/zg2i4DFjWNf2Pst

3 IMPLEMENTATIE

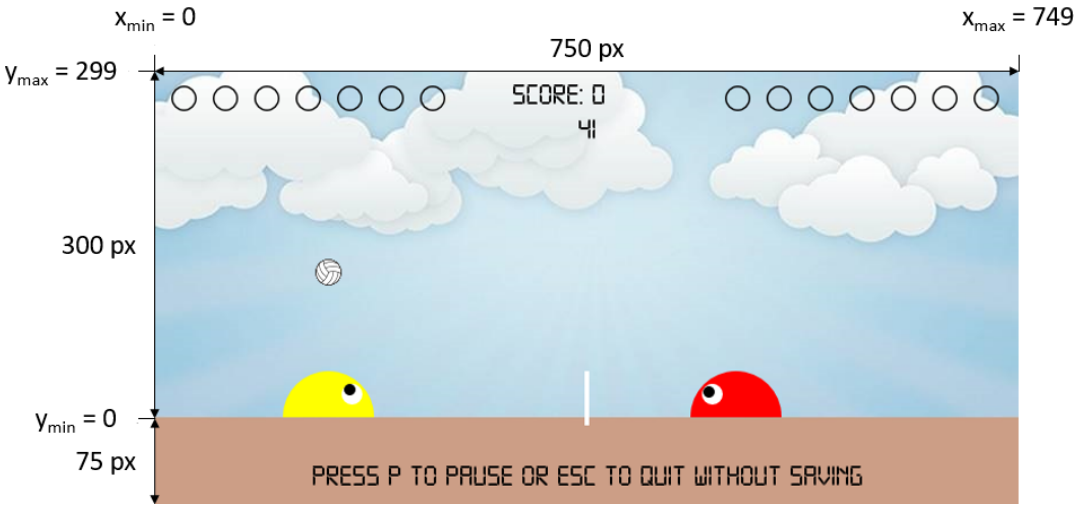
3.1 Het coördinatenstelsel

Aangezien Slime Volley een eenvoudig tweedimensionaal spel is, kan elk object op het scherm voorgesteld worden door een stel van (x, y) coördinaten. Het gebruikte coördinatensysteem van Allegro is equivalent aan dat van Java Swing, waarbij de x-coördinaat toeneemt naar rechts en de y-coördinaat toeneemt naar beneden (zie Figuur 2).



Figuur 2: Visualisatie van het Allegro assenstelsel.

Omdat er in het spel een zekere hoeveelheid wiskunde en fysica aan bod komt, is het niet onlogisch om over te stappen op een klassiek assenstelsel met een toename van de x- en y-coördinaat naar rechts en naar boven respectievelijk. Dit vraagt uiteindelijk slechts een kleine aanpassing bij het weergeven van de entiteiten: y afbeelden op $y_{max} - y$. Om de logica verbeterbaar te houden, wordt de volgende afspraak gemaakt: wanneer een speler zich op de grond bevindt (i.e. niet in de lucht springt), geldt er dat $y = 0$. De verschillende afmetingen worden volledigheidshalve weergegeven in Figuur 3. Merk op dat $y_{min} = 0$ enkel slaat op het "werkvenster" waarin de slimes en de bal zich bewegen: er is nog een "vloer" van 75 px voorzien waarin instructies of berichten geplaatst kunnen worden.

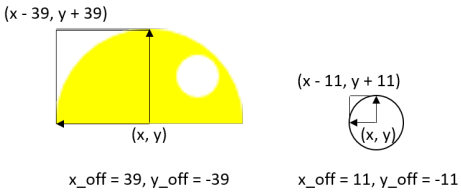


Figuur 3: Visualisatie van de afmetingen van het spel.

Bij het weergeven van afbeeldingen (bitmaps) in Allegro, worden de (x, y) coördinaten van de linkerbovenhoek gebruikt. Wanneer de coördinaten (0, 0) opgegeven worden, wordt de afbeelding in het Allegrostelsel in de linkerbovenhoek weergegeven. In ons klassieke assenstelsel, zou dit overeenkomen met de coördinaten (0, 299).

Voor de bewegende objecten in het spel (i.e. de bal en de slimes), is het echter makkelijker om te werken met de x- en y-coördinaten van het middelpunt van de (halve) cirkel. Op die manier hebben de bal en de gele slime in Figuur 3 dezelf-

de x-coördinaat, wat het maken van berekeningen vereenvoudigt. Gezien de grootte van de meegegeven afbeeldingen (cfr. Tabel 3), komt dit overeen met een translatie van (39, -39) en (11, -11) voor de slimes en de bal respectievelijk, zie Figuur 4. Om rekening te houden met deze translatie, kan er gebruik gemaakt worden van de functie `void Graphics::DrawBitmap(Sprite sprite, float dx, float dy, float cx, float cy)`, zie Subsectie 3.10.5. Merk op dat de onderkant van een slime op deze manier inderdaad overeenkomt met een y-waarde van 0, en dat er nooit een x-waarde lager dan 39 of hoger dan 710 kan voorkomen.



Figuur 4: Translatie van de (x, y) coördinaten van een slime en de bal.

3.2 De klasse Game

De implementatie van de klasse Game werd reeds voorzien. Deze initialiseert de Allegro Library en laadt vervolgens de verschillende afbeeldingen (ook wel sprites genoemd), lettertypes en highscores in vanuit de map assets. Eenmaal de functie Run() opgeroepen wordt, wordt er een eenvoudige game loop opgestart waarin het startmenu wordt weergegeven. Door middel van selectie via de pijltjestoetsen, kan de gebruiker uit vier verschillende opties kiezen:

- Singleplayer: start een spel tegen de computer (klasse GameSingle)
- Multiplayer: start een spel met twee spelers (klasse GameMulti)
- Highscores: geef de vijf hoogst behaalde scores weer
- Quit: sluit het spel af

Wanneer de highscores gekozen worden, wordt een nieuw menu getoond met daarin de vijf hoogst behaalde scores, die elk geselecteerd kunnen worden om de replay van een spel te tonen (klasse GameReplay). Afhankelijk van de teruggegeven waarde van de Run()-functies van deze drie types spellen, zal een nieuw level of spel gestart worden, of wordt het startmenu opnieuw weergegeven. Elk van de klassen GameSingle, GameMulti en GameReplay zijn gebaseerd op een entity-component-system framework, waarover hieronder meer informatie volgt.

3.3 Entity-component-system framework

Meer informatie over een entity-component-system framework was te vinden in de eerder vermelde bronnen in Sectie 1.3. Voor dit project werd reeds een basisversie geïmplementeerd, die wordt weergegeven in Figuur 5. Het framework bestaat uit een aantal belangrijke klassen, die hieronder kort overlopen worden.

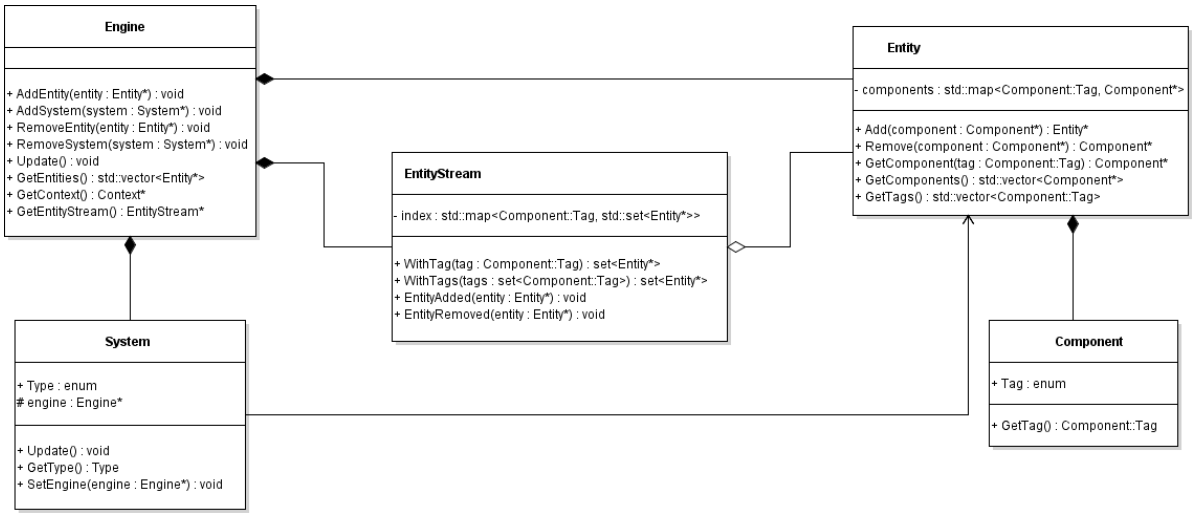
3.3.1 Component

Een Component is een heel eenvoudige data-container. De bedoeling is conceptuele eigenschappen in een klasse te steken die overerft van de klasse Component. Voor de eenvoud mogen de attributen in dit project publiek gehouden worden, aangezien deze dikwijls zullen moeten uitgelezen worden.

3.3.2 Entity

Een Entity stelt om het even welk object voor in het spel. Entiteiten worden opgevuld door een verzameling Components toe te voegen die de eigenschappen van het object voorstellen. De belangrijkste methoden van de klasse Entity zijn:

- `Entity* Add(Component* component)`: Voegt een Component toe via een pointer naar deze component. Een pointer naar de Entity wordt teruggegeven om via een builder pattern verder te kunnen werken (https://en.wikipedia.org/wiki/Builder_pattern), maar dit is geen verplichting;



Figuur 5: Het te implementeren entity-component-system framework.

- `Component* Remove(Component* component)`: Verwijdert een Component via een pointer naar de te verwijderen component. Een pointer naar de verwijderde Component wordt teruggegeven (verwijderen betekent niet vernietigen);
- `Component* GetComponent(Component::Tag tag)`: Geeft een pointer naar een Component terug door opgeven van zijn Tag (enum-waarde);
- `std::vector<Component*> GetComponents()`: Geeft een vector van Component pointers terug die wijzen naar alle Componenten van deze entiteit;
- `std::vector<Component::Tag> GetTags()`: Geeft een vector terug met alle Tags (enum-waarden) van de Componenten van deze entiteit.

3.3.3 System

Een System stelt een bepaald aspect van het spel voor, zoals beweging, collision detection en printen naar het scherm. Alle systemen zullen overerven van de hoofdklasse System, en zullen een of meerdere entiteiten verwerken in elke iteratie van de game loop. System beschikt over de volgende methoden:

- `virtual void Update()= 0`: Dit is een methode die verplicht te implementeren is door alle overervende System klassen, waarin alle logica van het systeem uitgevoerd wordt. Deze methode wordt eenmaal per game loop iteratie aangeroepen;
- `virtual Type GetType()= 0`: Dit is een methode die verplicht te implementeren is door alle overervende System klassen, die een unieke enum-waarde (Type) teruggeeft voor het desbetreffende systeem;
- `void SetEngine(Engine* _engine)`: Dit is een eenvoudige setter waarmee de protected variabele engine ingesteld wordt om een pointer naar de engine beschikbaar te hebben in alle overervende systemen.

3.3.4 Engine

De Engine is de belangrijkste klasse van het framework, waarin alle klassen functioneel samenkomen. Zowel Entities als Systems worden toegevoegd aan de engine. De Engine heeft een Update()-methode die zal opgeroepen worden voor elke iteratie van de game loop, waarin elk systeem wordt aangeroepen. Systems zullen via de EntityStream (zie Subsectie 3.3.5) de nodige entiteiten kunnen opvragen om deze te verwerken. Indien nodig kunnen systemen ook met andere systemen communiceren door ze op te vragen bij de Engine. De klasse Engine beschikt over de volgende methoden:

- `void AddEntity(Entity* entity)`: Voegt een Entity toe aan de Engine via een pointer;
- `void RemoveEntity(Entity* entity)`: Verwijdert een Entity van de Engine via een pointer;
- `void AddSystem(System* system)`: Voegt een System toe aan de Engine via een pointer;
- `void RemoveSystem(System* system)`: Verwijdert een System van de Engine via een pointer;
- `void Update()`: Roept om de beurt de Update()-methode aan op alle Systems die momenteel aan de Engine toegevoegd zijn;
- `std::vector<Entity*> GetEntities()`: Geeft een vector met pointers naar alle Entities uit de Engine terug;
- `Context* GetContext()`: Geeft een pointer naar de Context klasse terug (zie Subsectie 3.10.1);
- `EntityStream* GetEntityStream()`: Geeft een pointer terug naar het EntityStream object.

3.3.5 EntityStream

Aan de EntityStream kan een set van Entities opgevraagd worden door op te geven over welke Componenten ze moeten beschikken. Een RenderSystem zal alle Entities willen die beschikken over een Component van het type Sprite. EntityStream moet dus op de hoogte zijn van alle Entities die toegevoegd en/of verwijderd worden van de Engine. Dit is mogelijk door in de klasse Engine bij toevoegen of verwijderen van een Entity de EntityAdded en EntityRemoved-methoden van EntityStream op te roepen. Deze zullen de interne mapping in een `std::map` van (sleutel) Component::Tag naar (waarde) `std::set<Entity*>` aanpassen. EntityStream beschikt over de volgende methoden:

- `std::set<Entity*> WithTag(Component::Tag tag)`: Geeft een set terug met pointers naar de Entities die toegevoegd zijn aan de Engine én die een component met de opgegeven tag bevatten.
- `std::set<Entity*> WithTags(std::set<Component::Tag> tags)`: Geeft een set terug met pointers naar de Entities die toegevoegd zijn aan de Engine en over alle opgegeven componenten beschikken.
- `void EntityAdded(Entity* entity)`: Wordt aangeroepen door de Engine bij toevoegen van een Entity. Een pointer naar de Entity wordt aangeleverd via het argument. Daarmee wordt de interne mapping aangepast, zodat de WithTag(s)-methoden correct kunnen gebruikt worden.
- `void EntityRemoved(Entity* entity)`: Wordt aangeroepen door de Engine bij verwijderen van een Entity. Een pointer naar de entiteit wordt aangeleverd via het argument. Daarmee wordt de interne mapping aangepast, zodat de WithTag(s)-methoden correct kunnen gebruikt worden.

3.4 Componenten

De verschillende componenten hangen af van de verschillende functionaliteiten van het spel en de manier waarop erop ingespeeld wordt. In de voorbeeldoplossing werden de componenten in Tabel 1 gebruikt. De component sprite wordt gebruikt door alle entiteiten die een afbeelding nodig hebben (zoals de slimes, de bal, en het net). De component motion wordt toegekend aan alle entiteiten die kunnen bewegen, volgens een bepaalde snelheid en versnelling (zie Subsectie 3.5.4). De component player slaat op de twee slimes, waarvan de ogen bewegen op basis van waar de bal zich bevindt (zie Subsectie 3.5.6. De component point wordt gebruikt door entiteiten die de tussenstand tussen de twee spelers aanduiden, met een maximum van zeven punten elk. De component ball wordt gebruikt als tag, i.e. om de overeenkomstige entiteit gemakkelijk op te vragen.

Bij wijze van voorbeeld zou het creëren van een entiteit van het type ball er als volgt uit kunnen zien:

```
Entity* ball = new Entity();
ball->Add(new ComponentSprite(Graphics::SPRITE_BALL, 150, 11, 738, -11,
    150, 11, 288, 11));
ball->Add(new ComponentMotion(0, 0, 0, -1));
ball->Add(new ComponentBall());
```

```
engine.AddEntity(ball);
```

De bal zal initieel weergegeven worden in het punt (150, 150), en stilhangen in de lucht. Doordat er echter een valversnelling is, zal de verticale snelheid toenemen naar onder (negatieve versnelling in het klassieke assenstelsel) en zal de bal naar beneden vallen.

Component	Attributen	Verklaring
Sprite	sprite	Afbeelding van het object
	x	De x-waarde in het assenstelsel uit Figuur 3
	x_min	Minimale x-waarde
	x_max	Maximale x-waarde
	x_off	Offset x-waarde met betrekking tot Figuur 4 (i.e. 39 en 11)
	y	De y-waarde in het assenstelsel uit Figuur 3
	y_min	Minimale y-waarde
	y_max	Maximale y-waarde
	y_off	Offset y-waarde met betrekking tot Figuur 4 (i.e. -39 en -11)
Motion	v_x	Horizontale snelheid
	v_y	Verticale snelheid
	a_x	Horizontale versnelling
	a_y	Verticale versnelling
	j_y	Nieuwe verticale snelheid wanneer een object zijn minimale y-waarde bereikt
Player	player_id	ID van de speler (1 voor links, 2 voor rechts)
	radius	De straal van de slime
	pupil_x	De x-waarde van de pupil
	pupil_y	De y-waarde van de pupil
Point	player_id	ID van de speler waar het punt bij hoort
	point_id	ID van het gescoorde punt waar de component bij hoort
Ball	N/A	N/A (geen attributen nodig)

Tabel 1: Componenten van de voorbeeldoplossing.

3.5 Systemen

Ook de verschillende systemen hangen wederom af van de verschillende functionaliteiten van het spel. In dit project worden de onderstaande systemen onderscheiden, elk met een aantal constanten die in Tabel 3 gedefinieerd worden. Houd er rekening mee dat de volgorde van het toevoegen van systemen aan de Engine van belang is; zo zal het tekenen van het frame telkens de laatste stap van de iteratie zijn.

3.5.1 InputMulti

Dit systeem past de snelheid van de slimes aan naargelang de input van de gebruiker. Om respectievelijk naar links, rechts of omhoog te gaan, wordt de linkse slime bewogen met A, D en W voor Qwerty of Q, D en Z voor Azerty, de rechtse slime met de pijlen naar links, rechts en omhoog.

3.5.2 InputSingle

Idem als voor InputMulti, maar nu wordt enkel de linker slime bewogen met de pijlen naar links, rechts en omhoog.

3.5.3 AI

Dit systeem beweegt de rechtse slime op basis van een vooropgestelde logica. Omdat dit een belangrijke component is, wordt de logica in wat meer detail besproken in Subectie 3.9.

3.5.4 Motion

Dit systeem past de positie aan van alle entiteiten met de Component motion. In elke iteratie worden de horizontale en verticale snelheid (uitgedrukt in pixels per frame) aangepast volgens de gedefinieerde versnelling (uitgedrukt in pixels per frame^2), om vervolgens de nieuwe x- en y-waarde van de entiteit te berekenen.

3.5.5 Collision

Dit systeem controleert op mogelijke botsingen van entiteiten met elkaar of met de muur, de grond en het net. De volgende botsingen moeten opgelost worden:

- Een botsing tussen een speler en de muren of het net, met behulp van de variabelen x_min en x_max;
- Een botsing tussen een speler en de vloer, met behulp van de variabele y_min;
- Een botsing tussen de bal en de muren, met behulp van de variabelen x_min en x_max;
- Een botsing tussen de bal en het net;
- Een botsing tussen de bal en een speler, reeds geïmplementeerd wegens spelspecifiek (implementatiedetails hoeven niet bestudeerd te worden).

Merk op dat de positie en de afmetingen van het net beschreven staan in Tabel 3.

3.5.6 Eyes

Dit systeem plaatst de pupillen van een slime op de juiste plaats, gegeven de posities van de slime en de bal. Hiertoe kan de richtingscoëfficiënt berekend worden tussen het centrum van het oog en het centrum van de bal (deze eerste kan bepaald worden via een eenvoudige translatie van de (x, y)-coördinaten van de slime, cfr. Tabel 3), om vervolgens de coördinaten van de pupillen op de rechte te bepalen.

3.5.7 Output

In een singleplayer spel pusht dit systeem elk frame de nieuwe coördinaten van de twee slimes en de bal naar een lijst van Coordinates. Zodra het level afgelopen is, worden deze weggeschreven naar het bestand "./assets/highscores/{start_time}_{level}.txt", met start_time het aantal seconden sinds epoch en level het huidige level (beide gedefinieerd in de hulpklasse Context, cfr. Subsectie 3.10.1. Wanneer het spel net begonnen is, zullen de eerste vijf lijnen er bijvoorbeeld als volgt uitzien (x speler 1, y speler 1, x speler 2, y speler 2, x bal, y bal):

```
150 0 594 0 150 133.125
150 0 588 0 150 132.375
150 0 582 0 150 131.250
150 0 576 0 150 129.750
150 0 570 0 150 127.875
```

De linker slime blijft staan (kolom 1-2), terwijl de rechter slime naar links beweegt (kolom 3-4) om een betere ontvangstpositie aan te nemen. De bal beweegt verticaal naar beneden (kolom 5-6), vooraleer te botsen met de linkse slime. Deze bestanden zullen ingelezen worden om een replay mogelijk te maken van de vijf spellen met de hoogste score. Merk op dat coördinaten enkel bijgehouden mogen worden wanneer ze verschillend zijn van de voorgaande, en dus niet wanneer het spel bijvoorbeeld gepauzeerd is.

3.5.8 StateMulti

Dit systeem behandelt de toestand van het multiplayer spel wanneer het spel actief (niet gepauzeerd) is. Wanneer een punt gescoord wordt, wordt een teller gebruikt om het spel tijdelijk te freezen. Op deze manier is het visueel duidelijk dat er een punt gescoord werd, en dat de spelers best hun toetsen kunnen loslaten. Zodra deze korte periode voorbij is, worden de spelers teruggezet op hun beginpositie en wordt de opslag toegekend aan de speler die het laatste punt won.

Waarde	Betekenis
0	De bal is in het spel (default)
-1	De speler heeft een level gewonnen in een singleplayer game
-2	De speler heeft een level verloren in een singleplayer game
-3	De linkse speler heeft de wedstrijd gewonnen in een multiplayer game
-4	De rechtse speler heeft de wedstrijd gewonnen in een multiplayer game
-5, -6, -7	De linkse speler heeft een punt gewonnen
-8, -9, -10	De rechtse speler heeft een punt gewonnen
1	De speler wil doorgaan na een gewonnen level in een singleplayer game
2	De speler wil stoppen in een singleplayer game
3	De speler wil herspelen na verloren/laatste level in een singleplayer game

Tabel 2: Voorbeeldwaarden voor de variabele state in de klasse Context.

Wanneer een van de spelers echter zeven punten gescoord heeft, moet er een melding verschijnen (geactiveerd door het Render-systeem, zie Subsectie 3.5.12) en moet er gewacht worden op input van de gebruiker: een spatie om opnieuw te beginnen, ESC om terug te gaan naar het startmenu.

Een belangrijke variabele voor deze klasse is het gehele getal state in de klasse Context. Deze variabele zal namelijk bijhouden in welke toestand het spel zich bevindt, zodat het Render systeem de juiste boodschap naar het scherm kan wegschrijven (zie Subsectie 3.5.12) en de klasse GameMulti weet wat er van het spel verlangd wordt (i.e. opnieuw spelen of het spel beëindigen). In de voorbeeldoplossing werden de waarden zoals weergegeven in Tabel 2 gebruikt.

De waarden -5 tot en met -7 en -8 tot en met -10 worden hierbij willekeurig gegenereerd, om niet steeds dezelfde boodschap naar het scherm te schrijven wanneer een punt gescoord wordt. In de klasse SystemRender kan dan bijvoorbeeld het volgende gebruikt worden:

```
if (engine->GetContext()->GetState() == -5)
{
        if (engine->GetContext()->GetLevel() == 0)
        {
                Graphics::Instance().DrawString("Nice touch, Player 1!",
                    375, 310, c, Graphics::ALIGN_CENTER);
        }
        else
        {
                Graphics::Instance().DrawString("Nice touch!", 375, 310, c
                    , Graphics::ALIGN_CENTER);
        }
}
```

Het staat de student vrij om de gebruikte waarden voor deze variabele zelf te kiezen. Enkel de waarden 0 tot en met 3 moeten gerespecteerd worden, omdat deze gebruikt worden in de klasse Game.

3.5.9 StateSingle

Idem als voor StateMulti, maar nu voor het singleplayer spel. De mogelijke opties aan het einde van een spel hangen nu af van winst of verlies in het level (doorgaan naar het volgende level of opnieuw beginnen respectievelijk). Bovendien moet er voor elk punt voor of tegen de speler een update van de score gebeuren (+200 * level en -100 * level respectievelijk). Ook in deze klasse wordt de variabele state in de klasse Context aangepast waar nodig.

3.5.10 Replay

Dit systeem leest een bestand met coördinaten in en pusht deze naar een lijst van Coordinates. Afhankelijk van de afspeelsnelheid, die geregeld kan worden met de pijlen naar links en rechts, worden geen, een of meerdere frames uit de lijst verwijderd en worden de posities van de slimes en de bal aangepast. Net zoals in StateSingle moeten het aantal gescoorde

punten en de score aangepast worden wanneer een punt gemaakt wordt. Met behulp van de pijl naar boven moet bovendien geskipt worden naar het volgende punt (i.e. vanaf de opslag), met behulp van de pijl naar onder naar het volgende level (als er een is). Deze laatste twee functionaliteiten kunnen zeer eenvoudig voorzien worden, zoals aangetoond in de reeds geïmplementeerde methoden GoToNextPoint() en GoToNextLevel(). Merk op dat de replay level per level gebeurt: zodra een GameReplay afgewerkt is, zal een nieuwe opgestart worden als er een volgend level bestaat. Ook in deze klasse wordt de variabele state in de klasse Context aangepast waar nodig.

3.5.11 Points

Dit systeem itereert over alle entiteiten van het type Point, en past de afbeelding aan van een lege naar een ingekleurde cirkel voor elk gescoord punt.

3.5.12 Render

Het systeem Render zal voor elk frame de weergave op het scherm voor zich nemen. Wanneer er getekend wordt, moet eerst en vooral het scherm leeggemaakt worden met `Graphics::Instance().ClearScreen();`, zodat op een leeg, zwart canvas kan begonnen worden met tekenen. Vervolgens worden alle entiteiten met component Sprite getekend, samen met de pupillen voor beide spelers. Ook de score en de bekomen waarde voor de frame rate worden naar het scherm geprint, samen met de instructies voor het pauzeren of afsluiten van het spel. Deze klasse maakt gebruik van de variabele state in de klasse Context: wanneer een punt gemaakt werd, wordt feedback gegeven naar de gebruiker (zoals “Nice point, Player 1!”) en wanneer een level of spel afgelopen is, worden de juiste instructies getoond (zoals “Press space to continue to next level or ESC to quit”). Als laatste stap wordt `Graphics::Instance().ExecuteDraws();` aangeroepen. Elke tekenoperatie wordt namelijk naar een bitmap in het geheugen geschreven, en bij oproepen van ExecuteDraws() wordt de bitmap in het geheugen gewisseld met de bitmap op het scherm. Deze techniek, beter gekend als buffering, zorgt ervoor dat er tijdens het renderen geen flikkering in het scherm ontstaat.

3.6 De klasse GameMulti

Deze klasse voorziet een multiplayer spel. Bij initialisatie worden de benodigde systemen toegevoegd via de AddSystems()-methode en worden de nodige entiteiten aangemaakt in de MakeEntities()-methode. De systemen werden reeds voorzien in game_multi.h, de entiteiten zijn zelf nog toe te voegen.

In de game loop in de Run()-methode wordt er telkens een nieuwe iteratie gestart via de AllegroLib singleton. Hierna moet het huidige Allegro event opgevraagd worden, zodat de juiste actie ondernomen kan worden. Wanneer het een event betreft dat wijst op het indrukken van een toets op het toetsenbord, wordt de overeenkomstige boolean getoggled in de hulpklasse Context (zie Sectie 3.10). Wanneer de toets losgelaten wordt, wordt deze geüntoggled. Wanneer het event wijst op het verstrijken van de game state timer (het spel zelf moet geüpdatet worden), wordt een update van de engine uitgevoerd. Merk op dat er op dit moment nog een aantal zaken aangesproken worden via de Graphics klasse, om een statisch beeld weer te kunnen geven. Deze code moet uiteraard uitgevoerd worden door het Render systeem, zodat de klasse Graphics enkel van hieruit aangesproken zal worden.

Wanneer het spel gedaan is, worden de systemen verwijderd en worden alle aangemaakte entiteiten vernietigd met behulp van de RemoveSystems() en RemoveEntities()-methoden.

3.7 De klasse GameSingle

De implementatie van deze klasse is vrij gelijkaardig aan die van de klasse GameMulti. Er zullen een aantal extra systemen toegevoegd moeten worden, en wat de entiteiten betreft zal de sprite van de tegenstander afhangen van het huidige level. De belangrijkste wijziging is dat de AI ingevoegd moet worden, die besproken wordt in Sectie 3.9. Merk op dat de Run()-methode nu de variabele state uit de klasse Context als een geheel getal teruggeeft, dat afhankelijk is van het verloop van het spel en de beslissing van de speler:

- De speler won het level en wil naar het volgende level: 1
- De speler won het level en wil of kan (einde spel) niet naar het volgende level: 2

- De speler verloor het level en wil niet opnieuw beginnen: 2
- De speler verloor het level en wil opnieuw beginnen: 3

De teruggegeven waarde wordt gebruikt door de klasse Game, om te beslissen wat er na het beëindigen van het single-player spel moet gebeuren.

3.8 De klasse GameReplay

De implementatie van deze klasse is gelijkaardig aan de twee bovenstaande. Tracht zelf te achterhalen wat er verwacht wordt in welke situatie.

3.9 De AI

De AI is de logische component die de tegenstander zal aansturen bij het spelen van een singleplayer spel. In de klasse SystemAI wordt in pseudocode uitgelegd wat er verwacht wordt van de tegenstander in levels 1 en 2. Een derde level kan als uitbreiding geïmplementeerd worden, maar dit is geen verplichting. Hieronder wordt kort de flow omschreven van wat er gebeurt in levels 1 en 2, al zal veel pas duidelijker worden bij het doornemen van de pseudocode.

3.9.1 Level 1

De logica is opgesplitst in vier delen: de opslag, de positionering, het springen en de return. In level 1 is er maar één enkele opslag, waarbij er in een enkele stap naar rechts gegaan en gesprongen wordt. Wanneer de bal in het spel is en in het linkervak lijkt te vallen, tracht de slime een basispositie aan te nemen. Wanneer de bal op het eigen veld lijkt te vallen, zijn er een aantal scenario's waarin de slime zal springen (bijvoorbeeld wanneer de bal van uiterst rechts over het net gespeeld moet worden, en er dus extra snelheid nodig is). Daarnaast kan de slime ook naar links of rechts bewegen om de bal beter over te kunnen spelen, wat in bepaalde gevallen nodig zal zijn. Om gefundeerde beslissingen te nemen, zal de functie `XBallBelow(double y_target)` gebruikt worden om te bepalen op welke positie de bal zich zal bevinden wanneer zijn verticale positie voor het eerst lager wordt dan y_target. Op basis van deze positie bepaalt de slime of hij een bepaalde beweging moet inzetten naar de bal. Merk op dat er een variabele state gebruikt wordt om een opslag te geven. Deze variabele houdt bij waar in het punt de slime zich bevindt, bijvoorbeeld om te weten wanneer hij bepaalde acties moet doen die tot een geslaagde opslag leiden.

3.9.2 Level 2

In level 2 worden er twee welbepaalde nieuwe opslagen ingevoerd en kiest de slime een betere basispositie uit om aces kort achter het net te vermijden. Verder is de implementatie vrij gelijkaardig aan level 1.

3.9.3 Level 3

Probeer, als uitbreiding, een moeilijker level te maken. Denk hierbij aan nieuwe opslagen, een betere basispositie, een betere definitie van wanneer een bal in het eigen veld botst en nauwer gedefinieerde gevallen waarin er gesprongen mag worden.

3.10 Hulpklassen

3.10.1 Context

Dit is een hulpklasse waar een aantal variabelen in staan die aangepast en opgevraagd kunnen worden. Het bijhouden van de overkoepelende game state is daar slechts één voorbeeld van. Merk op dat, aangezien Context meegegeven wordt aan een Game en vervolgens aan een Engine, de variabele context via de engine opgevraagd kan worden in elk systeem.

3.10.2 Color

Dit is een hulpklasse die toelaat een kleur voor te stellen, ofwel als RGB met waarden van 0-255, ofwel als RGBA met waarden van 0-1.

3.10.3 Tags

Dit is een hulpklasse die toelaat om aan de hand van een builder pattern een set van Compontent::Tag tags op te stellen. Dit kan handig zijn bij het opvragen van Entities aan de EntityStream klasse, door bijvoorbeeld het commando `std::set<Component::Tags> tags = (Component::Sprite).And(Component::Motion).List();` te gebruiken om alle entiteiten te verkrijgen die bewegen.

3.10.4 AllegroLib

AllegroLib is een wrapper rond de Allegro bibliotheek. Allegro laat toe om op relatief eenvoudige wijze naar het scherm te tekenen en een game te ontwikkelen. Via AllegroLib werd de library nog sterk vereenvoudigd, en het is dan ook de bedoeling dat er gebruikt gemaakt wordt van deze wrapper waar mogelijk. Allegro-specifieke methoden, die typisch beginnen met "al_" moeten dus niet gebruikt worden. De volgende methoden werden voorzien:

- `void Init(int _screen_width, int _screen_height, float _fps)`: Initialiseert de Allegro bibliotheek;
- `void StartLoop()`: Deze methode start de achterliggende timer die 40 keer per seconde de game loop laat uitvoeren. Ze wordt dus opgeroepen net voor het starten van de game loop;
- `void StartIteration()`: Deze methode wacht totdat het volgende Allegro event ontvangen wordt. Ze wordt als eerste opgeroepen in een iteratie van de game loop;
- `void Destroy()`: Ruimt alle interne structuren van Allegro op, en wordt als laatste methode opgeroepen in het programma;
- `ALLEGRO_EVENT GetCurrentEvent()`: Staat toe om het laatste Allegro event op te vragen en te verwerken. Dit kan bijvoorbeeld gebruikt worden om toetsenbord events af te handelen;
- `bool IsWindowClosed()`: Geeft aan dat het laatste opgevangen Allegro event al dan niet een window close event was en dat dus het scherm werd gesloten door middel van het kruisje rechts boven;
- `bool IsTimerEvent()`: Geeft aan dat het laatste opgevangen Allegro event al dan niet een timer event was;
- `void ShowError(string msg)`: Print een foutmelding msg naar het scherm.

Merk op dat AllegroLib als singleton werd geïmplementeerd. Opvragen van de instance doe je dus steeds met de statische methode `AllegroLib::Instance()`, waarop de nodige methoden opgeroepen kunnen worden.

3.10.5 Graphics

De klasse Graphics voorziet een aantal methoden om verschillende fonttypes en afbeeldingen in te laden, en om tekst en afbeeldingen weer te geven op het scherm. Ook deze klasse werd als singleton geïmplementeerd. De volgende methoden werden voorzien:

- `void LoadFonts()`: De fonts worden ingeladen en opgeslagen in een lokale variabele;
- `void LoadSpriteCache()`: De sprites worden ingeladen in een vector van ALLEGRO_BITMAP*, die als cache gebruikt kan worden. De vector is geïndexeerd door middel van een enum met de namen van de sprites;
- `void UnLoadFonts()`: Verwijder de ingeladen fonts;
- `void UnLoadSpriteCache()`: Verwijder de ingeladen sprites;
- `void ExecuteDraws()`: Wissel de bitmap in het geheugen met die op het scherm;

Verklaring	Waarde
Afmetingen van de bal	23 x 23 (straal 12)
Verticale startpositie van de bal	133.5
Horizontale versnelling van de bal	0
Verticale versnelling van de bal	-0.375
Afmetingen van de slimes	79 x 40 (straal 40)
Horizontale startpositie van de linkse slime	150
Horizontale startpositie van de rechtse slime	600
Horizontale versnelling van de slimes	0
Verticale versnelling van de slimes	-0.750
Snelheid naar links of rechts wanneer de slime beweegt	6
Initiële snelheid naar boven wanneer de slime omhoog springt	11.625
Offset van het centrum van het oog voor de linkse slime	(20, 20)
Offset van het centrum van het oog voor de rechtse slime	(-20, 20)
Afmetingen van het net	4 x 47
Coördinaten van het net	(373, 39)
Freeze time na het scoren van een punt [s]	1.2
Aantal te scoren punten per spel of level	7
Minimale afspeelsnelheid replay	0.5
Maximale afspeelsnelheid replay	4

Tabel 3: Constanten in het spel.

- `void ClearScreen()`: Leeg het scherm door het zwart te schilderen;
- `void SetBackground(Sprite sprite)`: Stel de achtergrondafbeelding in;
- `void DrawBackground()`: Teken de huidige achtergrondafbeelding;
- `void DrawBitmap(Sprite sprite, float dx, float dy, float cx, float cy)`: Teken de sprite, waarbij de coördinaten dx en dy overeenkomen met de offset ten opzichte van de linkerbovenhoek van het scherm, en de coördinaten cx en cy overeenkomen met een offset ten opzichte van de linkerbovenhoek van de afbeelding;
- `void DrawString(string str, float dx, float dy, Color c, Align align)`: Teken een string op positie (dx, dy) in de kleur c en met de opgegeven uitlijning.

3.11 Constanten

Tabel 3 definieert de constanten die gehanteerd worden in dit project. Alle maten van snelheid en versnelling worden uitgedrukt in de eenheid "per frame". De header Constants.h kan ingeladen worden om deze variabelen eenvoudiger aan te passen. Merk op dat het toegelaten en zelfs aangeraden is dit bestand verder uit te breiden, zodat aanpassingen eenvoudiger uitgevoerd kunnen worden.

4 BESTANDEN

Een deel van de code wordt bij de opgave meegegeven. Deze code is te vinden in de repository die voor elke groep reeds klaargezet is. Het is de bedoeling de bestanden waar nodig aan te passen om zo tot een werkende oplossing te komen. De onderstaande bestanden zijn geheel of gedeeltelijk gegeven.

4.1 Headerbestanden

De headerbestanden omvatten onder meer de verschillende componenten en systems, zoals reeds besproken in Sectie 3. Het wordt aangeraden de signatuur zoveel als mogelijk te behouden, al is het toegestaan om kleine aanpassingen te doen wanneer nodig. Zorg er wel voor dat dit duidelijk is door het nodige commentaar bij extra functies en variabelen te voorzien.

4.2 Implementatiebestanden

De implementatiebestanden omvatten de logica voor het spel. De bestanden main.cpp, allegro_lib.cpp, graphics.cpp, game.cpp, entity.cpp, entity_stream.cpp, engine.cpp en context.cpp werden reeds volledig geïmplementeerd, alle andere moeten nog worden aangevuld.

4.3 Assets

De folders fonts en images bevatten de benodigde lettertypen en afbeeldingen om het spel te spelen. De folder highscores bevat een bestand highscores.txt en een eerste highscore, maar zal dus later uitgebreid worden met meerdere highscore bestanden die de coördinaten van de bewegende entiteiten in een singleplayer spel vastleggen.

5 COMPILEREN EN UITVOEREN

Let op dat Windows en Mac OSX gebruikers de PGM virtuele machine[7] dienen te gebruiken om het project te kunnen ontwikkelen. Voor alle duidelijkheid: WSL wordt niet gebruikt voor het project.

Enkel de Allegro libary moeten jullie nog toevoegen en dit kan op de volgende manier:

```
sudo add-apt-repository ppa:allegro/5.2
sudo apt install liballegro5-dev
```

De volledige installatiehandleiding voor het project is ter referentie te vinden in de algemene documentatie repository van het vak Programmeren[8].

[7] https://github.ugent.be/Programmeren/docs/blob/master/setup-vm.md
[8] https://github.ugent.be/Programmeren/docs/blob/master/setup-project.md

6 STAPPENPLAN

Hieronder wordt een voorstel gegeven voor een stappenplan voor dit project. Het is niet verplicht dit plan te volgen, maar het kan een leidraad zijn om het project tot een goed einde te brengen. Er wordt aangeraden eerst de multiplayer te maken, vervolgens de singleplayer en ten slotte de replay. Wat het entity-component-system framework betreft, moeten enkel de spelspecifieke systemen nog geïmplementeerd worden. Om dit zo overzichtelijk mogelijk te houden, wordt er best systeem per systeem gewerkt, zoals hieronder weergegeven.

1. Controleer dat de opgavecode compileert en uitvoerbaar is. Als alles juist is, zorgt het uitvoeren van de executable ervoor dat het startmenu wordt weergegeven.
2. Implementeer de nodige functies in de klasse GameMulti. Maak één entiteit aan met de component Bal, twee entiteiten met de component Player, veertien entiteiten met de component Point (tweemaal zeven, het maximum aantal punten per speler) en het net met enkel de component Sprite. Implementeer een eerste versie van de klasse SystemRender, die in elke iteratie alle entities met een component Sprite op de juiste plaats weergeeft. Wanneer dezelfde coördinaten gebruikt worden als in het voorbeeld (tijdelijk in elke game loop), zal een gelijkaardige statische afbeelding weergegeven worden als in stap 1.
3. Implementeer de Update()-functie in de klasse SystemMotion, voeg het systeem toe in de klasse GameMulti en voeg een component Motion toe aan de bal en slimes. Zowel de bal als de slimes zouden verticaal naar beneden moeten vallen bij het opstarten van het spel.
4. Implementeer de nodige functies in de klasse SystemCollision, die er onder meer voor zal zorgen dat een slime zijn grenzen respecteert (i.e. op zijn helft van het veld blijft en geen y-waarde onder 0 kan hebben) en dat de bal in het spel blijft. Voeg het systeem toe in de klasse GameMulti en start het spel opnieuw op. De bal zou verticaal moeten blijven botsen op de linker slime.
5. Implementeer de Update()-functie in de klasse SystemInputMulti, die het al dan niet ingeduwd zijn van bepaalde toetsen omzet in een snelheid naar links, rechts of onder. Voeg het systeem toe in de klasse GameMulti, en het zou mogelijk moeten zijn de slimes te besturen.
6. Implementeer de Update()-functie in de klasse SystemStateMulti, die detecteert wanneer de bal de grond raakt en vervolgens de nodige acties onderneemt. Voeg het systeem toe in de klasse GameMulti, en het zou mogelijk moeten zijn om een vroege versie van het spel te spelen. Pas meteen ook de klasse SystemRender aan om, afhankelijk van de huidige game state, een boodschap naar het scherm te printen.
7. Implementeer de Update()-functie in de klasse SystemPoints, die de sprites van de punten aanpast aan de huidige score tussen de twee spelers. Voeg het systeem toe in de klasse GameMulti, en het zou mogelijk moeten zijn om de voortgang van het spel visueel te volgen.
8. Implementeer de Update()-functie in de klasse SystemEyes, die de pupillen van de ogen op de juiste plaats zet voor elke slime. Voeg het systeem toe in de klasse GameMulti, en het zou mogelijk moeten zijn om een volledige versie van het multiplayer spel te spelen.
9. Doe hetzelfde voor de klasse GameSingle als in stap 3, maar houd rekening met de verschillende entities per level.
10. Implementeer de Update-functies in de klassen SystemInputSingle en SystemStateSingle en voeg alle benodigde systemen toe aan de klasse GameSingle. Het zou mogelijk moeten zijn om een singleplayer spel te spelen tegen een niet-bewegende tegenstander.
11. Implementeer een eerste versie van de AI in de klasse SystemAI, gebruik makende van de opgegeven pseudocode. Voeg het systeem toe en werk de implementatie stap voor stap bij. Het eindresultaat is een werkende singleplayer met twee levels en nodige feedback op het scherm.
12. Implementeer de Update()-functie in de klasse SystemOutput, die de (x, y) coördinaten van de twee slimes en de bal wegschrijft naar een outputbestand zodra een level voorbij is.
13. Implementeer de klasse SystemReplay, die een inputbestand inleest naar een lijst van Coordinates en de positie van de drie bewegende entiteiten frame per frame aanpast. Het eindresultaat is een werkende replay van de spellen

in de highscores, waarbij de nodige functionaliteiten (trager/sneller afspelen, doorgaan naar het volgende punt of level) aanwezig zijn. **Wanneer deze versie van het spel correct gecompileerd kan worden en er geen memory leaks aanwezig zijn, voldoet je implementatie aan de voornaamste criteria van dit project.**

14. Implementeer vrijblijvend een aantal uitbreidingen op het spel. Mogelijkheden zijn een extra level in de single-player, het gebruik van meerdere slimes of ballen en het toepassen van nieuwe gravitatiewetten.

7 TIPS EN OPMERKINGEN

Hieronder worden een aantal tips en opmerkingen gegeven waar best rekening mee gehouden wordt in dit project:

- Weet dat de basis implementatie zoals deze gevraagd wordt in deze opgave genoeg is om het maximum van de punten te halen. Uitbreidingen dienen dus slechts toegevoegd te worden eens alles werkt en enkel indien je dit zelf zou willen. Uitbreidingen zullen geen extra punten met zich meebrengen.
- Aangezien dit project met twee studenten gemaakt dient te worden, zal een vlotte samenwerking noodzakelijk zijn. Zoals eerder vermeld, zal er reeds een repository voor elke groep beschikbaar zijn. Het gebruik van een andere repository is niet toegestaan. Het publiek plaatsen van de code zal gedetecteerd worden door onze interne monitoring tools en zal leiden tot puntenverlies.
- Begin niet onmiddellijk te programmeren. Lees eerst aandachtig de opgave en denk na hoe je de verschillende problemen zou oplossen.
- Het voordeel van een entity-component-system framework is dat je bepaalde systemen kan uitschakelen tijdens het debuggen, door ze tijdelijk niet aan de Engine toe te voegen.
- Maak zoveel mogelijk gebruik van de Standard Template Library (STL) (http://www.cppreference.com/wiki/stl/start).
- Alle opmerkingen bij de practica gelden ook hier: wees zuinig met geheugen, let op voor dangling pointers, geef alle gealloceerde geheugen ook weer vrij, controleer of bestanden wel correct geopend en gesloten worden, etc.
- Als programmeerstijl werd de Google C++ Style Guide gebruikt (https://google.github.io/styleguide/cppguide.html). Vele andere opties waren echter mogelijk, één enkel 'juist' alternatief bestaat niet. We vragen wel om, in de kader van dit project, de gehanteerde stijl zoveel mogelijk te respecteren.
- Het gebruik van C++11 / C++17 is toegestaan.
- Probeer compiler warnings te vermijden; deze duiden meestal op fouten die gemakkelijk op te lossen zijn.
- Vanzelfsprekend zijn er zaken waar deze opgave je vrij in laat; deze zijn naar eigen inzicht in te vullen. **Verduidelijk in elk geval je broncode met commentaar waar dit nuttig is**.
- Vragen over het project kunnen gesteld worden via (1) het mailadres pgm@lists.ugent.be en (2) Github issues. Wij trachten alle vragen zo snel mogelijk te beantwoorden. Indien er een probleem is met de samenwerking tussen de groepsleden, gelieve dit dan zo snel mogelijk door te geven, zodat er een oplossing uitgewerkt kan worden.

8 INDIENEN

Het project wordt gemaakt in groepen van twee studenten. Om in te dienen zorg je ervoor dat de volgende twee zaken uitgevoerd zijn ten laatste op zondag 16 mei 2021, 23u59:

1. De code staat klaar in de Github repository van jullie groep. De code van de master branch zal opgehaald worden.
2. Een verslag van het type PDF met een maximale lengte van twee pagina's is ingediend via de Ufora opdracht "Verslag Project". Het verslag omvat de volgende zaken:
    - De naam en richting van de groepsleden
    - Een korte uitleg over de belangrijkste ontwerpbeslissingen
    - Bij het niet (correct) uitvoeren van het programma: waarom?
    - De taakverdeling: wie heeft wat gedaan?

Als minimumvoorwaarde wordt vereist dat het project compileerbaar is, zoniet wordt een nulscore toegekend. Wanneer het programma niet correct uitgevoerd kan worden, wordt er gevraagd hierover een korte toelichting te geven in het verslag.

9 MEMORY LEAKS DETECTEREN MET ADDRESS SANITIZER

Op Linux, Mac en in WSL is het heel eenvoudig om memory leaks en andere geheugenfouten zoals een “double free” op te sporen door middel van AddressSanitizer (ASan). Dit is een geheugen foutdetector origineel ontwikkeld door Google voor de Clang compiler die sinds versie 4.8 ook in GCC verwerkt zit.

ASan staat automatisch aan voor de labo's van PGM. Je kan die voor andere programma's zelf aanzetten door de flags “-fsanitize=address -fno-omit-frame-pointer” mee te geven met de compiler en de linker.

De compiler linkt alle geheugenoperaties zoals `malloc` en `free` door naar ASan, zodat deze alle geheugenoperaties kan bijhouden. Bij het afsluiten van het programma wordt er gekeken welk geheugen er aangevraagd is zonder het te verwijderen en rapporteert ASan alle memory leaks in het formaat dat hieronder te zien is.

```
=================================================================
==22505==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 20000 byte(s) in 1 object(s) allocated from:
    #0 0x7f9ad892a848 in __interceptor_malloc (/usr/lib/x86_64-linux-gnu/
       libasan.so.5+0xee848)
    #1 0x55b9aede4800 in graphics_init /sysprog/game/graphics/
       opengl_game_renderer.c:364
    #2 0x55b9aede4605 in graphics_alloc /sysprog/game/graphics/
       opengl_game_renderer.c:343
    #3 0x55b9aedb75ee in main /sysprog/puzzle_bots_part2_main.cpp:43
    #4 0x7f9ad6f32b96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so
       .6+0x21b96)
```

Deze stack trace toont aan welke code het geheugen aangemaakt heeft. Zoals je ziet worden alle geheugenoperaties uiteindelijk uitgevoerd door `libasan`, de geheugenchecker zelf. Je mag regel #0 dus overslaan. Het eigenlijke geheugenlek doet zich voor in de `graphics_init` functie in `opengl_game_renderer.c` op lijn 364. De rest van de stacktrace toont de volledige volgorde van oproepen: de `main` functie roept op lijn 43 de `graphics_alloc` functie op, die op lijn 343 de `graphics_init` functie oproept.

Bij het debuggen wordt een extern consolevenster gebruikt dat afsluit bij het stoppen van het programma. Hierdoor is het niet mogelijk om de memory leaks te zien na een debugsessie. Start een normale uitvoering met het programma door “Terminal > Run Build Task” uit te voeren (`ctrl-shift-b`) en “run (+ build)” uit te voeren. Dit zal het project eerst builden en dan uitvoeren. De output zie je in het terminalvenster van Visual Studio Code. Merk op dat de optie “start without debugging” de debugger toch opstart. Dit is een gekende fout in de C/C++ extensie van Visual Studio Code.