Inleiding Programmeren 2024-2025
Examentaak C++ - 1ste zittijd
Game

Tom Hofkens, Tim Apers

Deadline: 13 januari 2024

1 Belangrijke informatie

- Maak deze opdracht *individueel*. Samenwerken is oneerlijk ten opzichte van je medestudenten. Je wordt gecontroleerd en bij samenwerking zijn we zeer streng. Bij het gebruik van code op het internet, vermeld je de bron. Het spreekt voor zich dat een kort fragment kopiëren uit een tutorial toegelaten is, maar een volledige oplossing specifiek aan de opdracht niet. Er wordt gebruik gemaakt van software voor plagiaat controle waarbij ook wordt vergeleken met online beschikbare code fragmenten.

- Het gebruik van generatieve AI-tools om code te genereren, wordt afgeraden, maar is toegestaan. Opnieuw dien je in de commentaar bij jouw code aan te duiden welke delen van de code je zelf schreef. **Je mag enkel code indienen die je zelf volledig begrijpt en kan uitleggen.**

- Tijdens de examenperiode zullen alle studenten die 100% haalden op de examentaak inzending op INGinious moeten langskomen op het projectexamen. Het doel van dit gedeelte is nagaan of jij zelf de auteur bent van de code en of je de code goed begrijpt. Je zal daarom een paar kleine aanpassingen moeten doen in een korte tijdspanne. Enkel indien je 100% haalde, kan je dus het projectexamen afleggen. De score na het projectexamen is de score die meetelt voor je eindcijfer van het vak.

- Bij de beoordeling houden we ook rekening met de kwaliteit van de code. Let er op dat je niet onnodig code dupliceert; dat de complexiteit van elke functie beperkt is (bijvoorbeeld door hulp functies te definiëren); dat je variabelen en functies duidelijke namen geeft; dat je je houdt aan code conventies; dat je complexe code goed documenteert.

- Benodigdheden: C++ compiler, we werken met C++ 11 of hoger.

- De deadline is 13 januari 2025, 22u.

2 Omschrijving

In dit project gaan we een simpel spel implementeren waar de speler een vijand moet verslagen door een wapen op te pakken. Het spel, een top-down single-player game, bevat 3 kamers met elks een speler, een wapen, en de vijand. De moeilijkheid van dit project ligt voornamelijk op het juiste gebruik van memory management, het begrijpen van klas structuren, en het concept van polymorfisme.

Welke onderdelen correct geïmplementeerd werden, bepaalt het uiteindelijke resultaat. Naast de beoordeling van de verschillende onderdelen van de opdracht, wordt ook de programmeerstijl beoordeeld. Hiermee wordt het schrijven van commentaren, const correctness, modulair programmeren, het gebruiken van de juiste data- en controlestructuren en het schrijven van overzichtelijke code bedoeld. Een uitzonderlijk goede of uitzonderlijk zwakke programmeerstijl kan het eindresultaat tot maximaal 20% beïnvloeden.

3 Opdracht

Aangezien dit het eerste jaar is dat we een variatie van het Inleiding Programmeren project lanceren kan het zijn dat er onduidelijkheden ontstaan. Als dit het geval is, aarzel dan niet om de assistent (Tim Apers) of de docent (Tom Hofkens) aan te spreken voor extra toelichting; deze opdracht heeft tot slot als doel jouw programmeervaardigheden te testen. Voor dit project wordt reeds code beschikbaar gesteld. Deze code omvat de volgende bestanden:

- directory *resources* met verschillende png, txt, en jpg bestanden: dit zijn de afbeeldingen van de entiteiten en de configuratie van de kamers. **Pas deze bestanden NIET aan.** Bij de beoordeling van deze taak wordt de originele versie van deze bestanden gebruikt, dus elke wijziging gaat verloren.
- *main.cpp*: Dit bestand opent een SFML-venster dat gebruikt wordt om het spel weer te geven. In principe hoef je ook in dit bestand niets te wijzigen.
- Game.cpp, Game.h, Room.cpp, Room.h, Entity.cpp, Entity.h: deze bestanden gaan de kern vormen van jouw project. Jij hebt de volledige vrijheid over hoe je deze bestanden aanpast. Voeg const, virtual, constructors, member functies en variabelen, access specifiers (public/private/protected) ... toe waar je dat wenselijk of nodig acht.
- Je mag ook nieuwe bestanden toevoegen aan jouw project.
- *CMakeLists.txt*: CLion gebruikt dit bestand om te zien hoe de code gecompileerd en gelinkt moet worden. Kijk dit bestand even na en pas aan voor jouw besturingssysteem zoals beschreven staat in de commentaren.

3.1 Initialisatie (10%)

Eerst en vooral moeten we verschillende getter en setter functies definiëren. Hieronder een overzicht van welke je moet implementeren en in welke klasse deze zich bevinden:

- **Entity**: *getPosition* en *setPosition*: Zo kunnen we de positie van een entiteit (muur, vijand, etc.) opvragen en in de kamers plaatsen. De positie van een entiteit hebben we voor jou al gedefinieerd als een struct "Positie" die een x en y coördinaat bevat.
- **Player**: *getAttackPower* en *setAttackPower*: Aangezien we in de huidige versie van het spel een zwaard kunnen oppakken, betekent dit ook dat de speler een attack power zal bezitten. Als die hoog genoeg is (door het zwaard op te pakken) kan hij de vijand verslagen. Om deze features te implementeren hebben we dus een bijhorende getter en setter nodig, maar ook een private member die de attack power van de speler aangeeft.
- **Room**: *getEntities* en *addEntity*: Een kamer bevat al een datastructuur voor het opslagen van entiteiten, namelijk een vector van Entity pointers. Schrijf eenderzijds een getter om deze vector op te vragen en een functie, addEntity, om entiteiten aan de lijst toe te voegen.

3.2 Kamers inlezen (20%)

Nu dat onze belangrijkste getters en setters geïmplementeerd zijn kunnen we de kamers inlezen. Je vindt de kamer lay-out van het spel onder de map "resources" in de file "map.txt". De betekenis van elk karakter is als volgt:

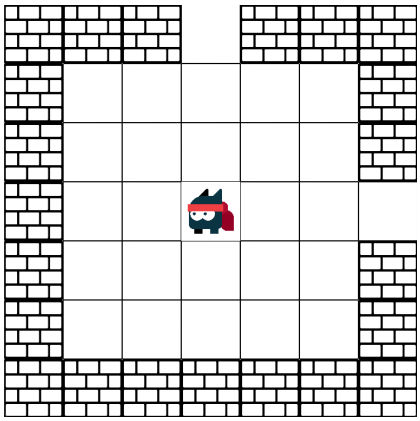
- # : Een muur (Wall)
- _ : Een vloer (Floor)
- % : Een vijand (Enemy)
- **!** : Een wapen (Weapon)
- **@** : Een speler (Player). Opgemerkt, onder de speler staat ook altijd een vloer!

Elke kamer heeft een afmeting van 700 x 700 wat dus ook betekent dat elke entiteit geplaatst moet worden op 100 units van elkaar. Bij grafische vormgeving ligt het punt (0, 0) frequent in de linkerbovenhoek en is de y-as gespiegeld (positief naar beneden, negatief naar boven). Men noemt dit ook wel een omgekeerd coördinatenstelsel en is ook het geval bij SFML. De logische coördinaten van een kamer in de linkerbovenhoek zien er daardoor als volgt uit:

$(0, 0)$	$(100, 0)$	$(200, 0)$	$(300, 0)$	$(400, 0)$	$(500, 0)$	$(600, 0)$
$(0, 100)$	$(100, 100)$	$(200, 100)$	$(300, 100)$	$(400, 100)$	$(500, 100)$	$(600, 100)$
$(0, 200)$	$(100, 200)$	$(200, 200)$	$(300, 200)$	$(400, 200)$	$(500, 200)$	$(600, 200)$
$(0, 300)$	$(100, 300)$	$(200, 300)$	$(300, 300)$	$(400, 300)$	$(500, 300)$	$(600, 300)$
$(0, 400)$	$(100, 400)$	$(200, 400)$	$(300, 400)$	$(400, 400)$	$(500, 400)$	$(600, 400)$
$(0, 500)$	$(100, 500)$	$(200, 500)$	$(300, 500)$	$(400, 500)$	$(500, 500)$	$(600, 500)$
$(0, 600)$	$(100, 600)$	$(200, 600)$	$(300, 600)$	$(400, 600)$	$(500, 600)$	$(600, 600)$

Rekening houdend met de hierboven vermelde details, vul de functie *loadMap* aan die de "map.txt" file inleest en de kamers initialiseert. Zorg er ook voor dat je de private member van **Game**, currentRoom, gelijk zet aan de kamer die de player bevat. Maak hiervoor nog geen gebruik van de *setCurrentRoom*, dit wordt later nog geïmplementeerd.

LET OP: De coördinaten van elke kamer zijn absoluut en niet relatief! Dit betekent dus dat de linkerbovenhoek van elke kamer overeenkomt met (0, 0), (0, 700), en (1300, 1300) respectievelijk. Als alles goed gaat en je heb de juiste current room, dan krijg je nu het volgende scherm te zien:

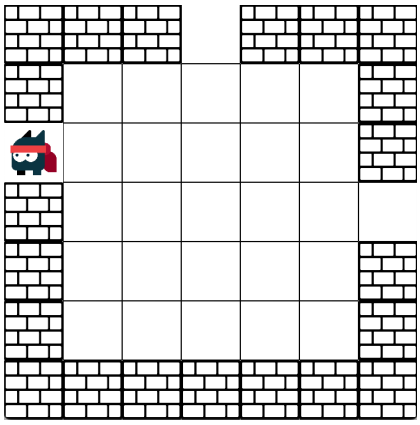


3.3 Beweging van de speler (10%)

Nu dat de kamers ingeladen kunnen worden is het tijd om de beweging van de speler te implementeren. Jullie doen dit door de functie *update* van de **Player** klasse te implementeren. Gelukkig hebben we voor jullie al een basis code gedefinieerd, het is aan jullie om deze verder aan te vullen. De commando's zijn als volgt:

- $\uparrow$: De speler verplaatst zich naar het vakje boven hem.
- $\leftarrow$: De speler verzet zich naar het vakje links van hem.
- $\downarrow$: De speler beweegt 1 vakje naar onder.
- $\rightarrow$: De speler beweegt een vakje naar rechts.

Na wat te rond te lopen kan je op volgende positie terechtkomen:



3.4 Collision handling (20%)

Zoals je zelf wel kunt zien is het momenteel mogelijk voor de speler om over muren te lopen. Om dit op te lossen moeten we een paar functies sequentieel aanvullen:

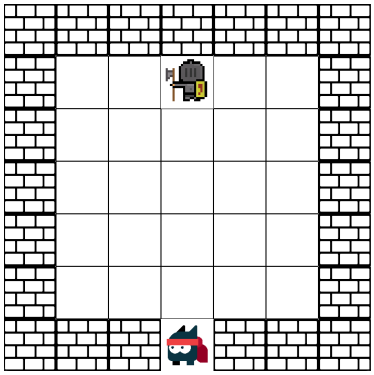
- **Entity**: *standsOn*: Dit is de eerste functie die we aanmaken, aangezien deze handig is voor het implementeren van de rest van het project. Deze functie geeft "true" terug als twee entiteiten op dezelfde positie staan en "false" als dat niet zo is.
- **Room**: *update*: Breid deze functie uit, zodat elke entiteit wordt gecheckt op een interactie met de speler. Een entiteit van een room kan interageren met de speler als deze op een entiteit staat. Je gebruikt voor deze implementatie dus de *standsOn* functie die je hiervoor gemaakt hebt. Een interactie wordt opgeroepen met de nog niet gedefinieerde *interacts* functie in de **Entity** klasse. Uiteindelijk zullen we voor elke entiteit een ander gedrag implementeren.
- **Wall**: *interacts*: Override deze functie van de **Entity** klasse. Om deze functie juist te implementeren, bedenk je een methode die ervoor zorgt dat een speler uit de muur wordt gezet waar hij opstaat. Daarnaast zal de speler ook terug naar de positie moeten gaan waar die vandaan kwam. Let dus goed op, deze functie zal vereisen dat je de huidige codebase hier en daar zal moeten aanvullen om deze functionaliteit te ondersteunen. Tot slot, return je de pointer naar de entiteit die potentieel verwijderd moet worden. In dit geval is dit niet van toepassing dus kan je gewoon een null pointer teruggeven.

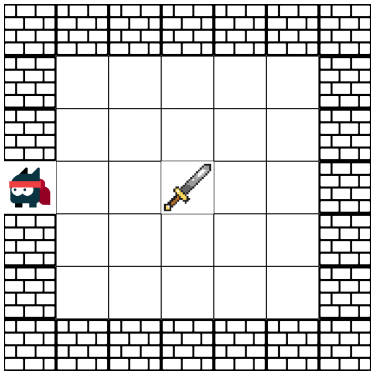
Als je al deze stappen juist doorlopen hebt, heb je op dit moment een spel waarbij je vrij kunt rondlopen, maar niet op muren.

3.5 Naar andere kamers gaan (10%)

Een andere belangrijke component van het spel is de mogelijkheid om van de ene kamer naar de andere te wandelen, wat op dit moment nog niet kan. Je doet dit door de functie *setCurrentRoom* aan te vullen in de klasse **Game**. Deze functie zal elke keer nadat een speler beweegt opgeroepen worden en moet uitzoeken in welke kamer de speler zich momenteel bevindt. Stel dat de speler zich nog altijd in dezelfde kamer bevindt, dan doe je niets. Anders, zet je de huidige kamer naar de nieuwe kamer.

Als deze functie juist geïmplementeerd is, is het momenteel mogelijk om je te verplaatsen naar de andere kamers:





3.6 Het spel finaliseren (30%)

Merk op dat je op de vijand en het zwaard kunt wandelen zonder dat er iets gebeurd. Om het spel te finaliseren, zullen we de interacties tussen de speler en deze entiteiten moeten definiëren. Implementeer volgende functies om het spel te beëindigen:

- **Weapon**: *interacts*: Als de speler op het wapen staat, dan verhoog je de attack power van de speler en geef je de pointer naar het wapen terug.
- **Enemy**: *interacts*: Staat de speler op de vijand, dan kijk je na wat de attack power van de speler is. Als deze niet hoog genoeg is (=0) dan gedraagt de vijand zich als een muur waar de speler niet op kan lopen en geef je een null pointer terug. Als de attack power wel hoog genoeg is (=1 nadat je het zwaard hebt opgepakt), dan geef je de pointer naar de vijand terug.

Je bent er bijna! Normaal gezien heb je nu een spel waarbij je niet op de vijand kan wandelen als je nog niet langs het zwaard bent gegaan. Als je eenmaal op het zwaard stapt heb je een hogere attack power waardoor het toegelaten is om op de vijand te wandelen. Echter, we willen ook graag dat het zwaard maar een keer kan opgepakt worden en dat als de vijand verslagen kan worden dat deze ook verdwijnt van het spelbord.

Breid de functie *update* in de klasse **Room** uit zodanig dat als de functie *interacts* een pointer teruggeeft die geen null pointer is, dan moet deze entiteit verwijderd worden van de room. Let in dit geval en ook in het algemeen op dat je geen memory leaks veroorzaakt! Als je dit correct geïmplementeerd hebt, is het project afgerond en heb je jouw eerste game geschreven.

4 Indienen

Dien de opdracht in op INGInious, onder 'IPExamens' > 'Examentaak C++ - 1ste zittijd: Game'. Je uploadt een zip bestand met alle .cpp en .h bestanden die je wijzigde. Dit is: Game.cpp, Game.h, Entity.cpp, Entity.h, Room.cpp, Room.h en alle bestanden die je zelf eventueel toevoegde. Voeg geen andere bestanden toe; zeker geen executables (.exe).