PACMAN GAME 2025/2026

dit is mijn report voor mijn PacMan indiening van Advanced Programming. De volgende pagina's ga ik alle onderdelen van mijn project uitleggen en en waarom ik dit zo heb gedaan.

Folder structuur

Mijn project is opgedeeld in 2 grote delen: App en Logic. De App is de complete representatie van Pacman, dit bepaalt dus al de input en output van het project en geeft dit door naar de Logic library. Het is perfect mogelijk om een eigen App te maken met een andere representatie (bv. Console of met joystick) en dit te koppelen.

De app library heeft ook de verschillende staten. Dit maakt het makkelijker om verschillende appilcatie situaties te maken en snel van deze te switchen en ook van elkaar te scheiden. Deze worden beheert door een StateManager die een stack heeft en zo de de verschillende staten beheert zonder rekening te houden met welke staat er actief is. De statemanager geeft ook alle belangerijke functies zoals (render of handleinput) door naar de actieve staat (bovenste op de stack).

Dit wordt dan allemaal beheert door de Game klasse. Dit is de klasse die alles op de juiste timing update en alle andere klasse aanmaakt. Dit geeft dan een update naar de Statemanager die dan ook de juiste functies called van de huidige actieve staat.

De Logic library is het deel van de code dat de logica van de pacman game bepaalt en handelt. Dit is dus de algoritmes voor alle entiteiten en onderdelen van het spel. Deze library kan je includen in je eigen project en apart compileren zonder de representatie. De logic library neemt dus dus inputs van de representatie en verwerkt deze, dan maakt de logic de aanpassingen die nodig zijn.

Designs

MenuState: dit de eerste staat dat je tegenkomt. Dit heeft de grote titel en een leaderboard met de top 5 score van deze game build. Hier kan je met ESC de game afsluiten of met ENTER doorgaan naar de gamestaat.

GameState: Dit is de algemene staat van de game. Hier wordt een Wereld klasse, Camera klasse, Factory klasse. Hiermee zullen dan de nodige views ontvangen worden door de factory en de coordinaten omgezet via de Camera klassen. Dit zorgt ervoor dat er makkelijker communicatie is vanuit de logic naar de representatie zodat er dan indien een gameover of er munten moeten verdwijnen snel via Events kan gehandeld worden of via een update functie.

PausedState: Dit is de staat waar je naartoe gaat als je ESC duwt op bij de GameState. Hier kan je pauseren aangezien de game niet verder gaat. Hier kan je terug naar de game gaan of terug naar het startmenu.

VictoryState: Je wordt naar deze state gepusht als de GameState te horen krijgt dat je een level hebt gecomplete. Hier krijg je je score te zien en kan je met een simpele ENTER naar het volgend level door een nieuwe Wereld klassen aan te maken en de juiste modifiers door te geven aan de constructor van de World. Dit zorgt ervoor dat alles mooi proper blijft en onnodige opbject van het vorig level verwijdert wordt.

GameOverState: Deze state is gelijkaardig aan de VictoryState maar is bedoelt om de user te tonen dat het spel gedaan is. Je kan alleen van deze state naar de menuState terug gaan en alle andere states waaronder de GameState worden verwijderd zodat er zeker een propere start kan gemaakt worden.

Model View: entiteiten zijn opgesplitst in een Model klasse en een view klasse. Dit zorgt ervoor dat dit gesplitst kan worden in de app en logic implementaties. Hierdoor is ook de UI implementatie gesplitst van de abstracte entiteiten die een genormaliseerde locatie en grote hebben. Dit zorgt ervoor dat de logic library zonder SFML kan gecompilerd worden en dat er makkelijke uitbereiding is.

MapLoader: Dit is een aparte klasse dat de map van de game inlaadt en verwerkt naar models die de world dan kan gebruiken. Door de virtual functies kunnen er meerdere klassen inheriten en andere input bestanden gebruikt worden om een map te laden zonder dat de World implementatie veranderd. Deze gebruikt dan ook Create”” functies van de World om makkelijk en abstract Models aan te maken zonder zelf aan de variable te komen.

World: in de World klasse wordt alles voor de logic bepaalt en geregeld. Hier heb ik voor de collision een visitor pattern gebruik om een visitor te sturen naar de entieiten die ik moet controlleren. Dit maakt dat alles mooi is verdeelt in functies en ik specifiekere gevallen kan maken. Daarnaast zorgt dit ook voor uitbereiding om meerdere object toe te voegen. Voor de movement van pacman bereken ik constant de het exact midden van de huidige tegel dit zorgt dat pacman vloeiend blijft bewegen. Daarnaast gebruik ik corner snapping met marges om hoeken vloeiender af te slagen. Voor de score gebruiker ik een observer patterns die aan de eetbare entiteiten vasthangt. Dit zorgt dat alles mooi gescheiden blijft en dat een klasse niet teveel opzich moet nemen.

Ghost & AI:

DirectChaseStrategy: Berekent de kortste route naar Pacman door de *Manhattan Distance* te minimaliseren. Het spook kiest op elk kruispunt de richting die hem dichter bij Pacman brengt.

FrontChaseStrategy: Een slimmere variant die probeert de weg af te snijden. Het doel is niet Pacman zelf, maar een tegel 3 stappen *voor* Pacman (afhankelijk van zijn kijkrichting).

FrightenedStrategy: Wordt actief als Pacman een speciale appel eet. De logica keert om: het spook kiest de richting die de afstand tot Pacman *maximaliseert*.

LockedStrategy: Simuleert een minder voorspelbaar patroon. Op kruispunten heeft het spook 50% kans om zijn huidige richting te behouden ("locked") en 50% kans om een willekeurige nieuwe richting te kiezen.