Multidisciplinair ingenieursproject 2017-2018

Leen Brouns, Veerle Ongenae

Team 9

Aleandro Delys
Qing Lee
Roel Moeyersoons
Lucas Van de Velde
Arthur Van Pellicom

UNIVERSITEIT GENT

Voorwoord

Voor dit project hebben Aleandro Delys, Qing Lee, Roel Moeyersoons, Lucas Van de Velde en Arthur Van Pellicom het iconische spel Pac-Man nagebouwd en uitgebreid. Het spel werd gemaakt voor het opleidingsonderdeel Multidisciplinair ingenieursproject informatica aan de Universiteit Gent. Hierbij moest er een niet-triviaal algoritme toegepast worden. Er werd gekozen voor het A* algoritme[9]. Het spel werd objectgeoriënteerd geprogrammeerd in Python 3. * m.b.v. het vrije cross-platform bibliotheek Pygame[7]. Tijdens het programmeren werd het project gedeeld via Github, een website die het versiebeheersysteem Git gebruikt. In dit verslag worden de belangrijkste geprogrammeerde klassen verduidelijkt.

Inhoudstafel

1. Inleiding

Om het spel goed uit te kunnen leggen, wordt er een ronde gedaan door het klassendiagram (Bijlage 1) en wordt er uitgelegd hoe er te werk is gegaan tijdens het ontwerpen van de klassen. Eveneens zullen kort enkele functionaliteiten en het geïmplementeerde algoritme besproken worden.

Voor een algemene uitleg van de methodes wordt er verwezen naar de documentatie, waarin alle methodes met hun parameters en een korte beschrijving staan - het is dus handig om deze ernaast te hebben tijdens het lezen van dit document. Deze documentatie is te vinden op de Github-pagina van Team-9 of via de site, zie referentie 13.

De ronde van het klassendiagram begint bij de klasse Game dat centraal op het klassendiagram staat (Bijlage 1).

1. Game

Het hart van het spel is de klasse Game, bij het opstarten wordt hiervan een object gemaakt die de grootte van het speelscherm, het doolhof en resolutie vastlegt bij constructie. Vanaf de opstart van het programma is het de bedoeling dat het Game-object een lus uitvoert die volgende zaken moet voorzien:

- Het bepalen in welke toestand het spel zich bevindt
- De mogelijkheid om het spel te pauzeren
- De mogelijkheid om het spel te stoppen
- Het hertekenen van het scherm

2. GameHandler

De GameHandler is een methode binnen de klasse Game, die instaat als “schakelaar” tussen de verschillende game-toestanden. Tijdens de game lus zijn er 6 mogelijke toestanden:

2.1: Startscherm
2.2: Ready-scherm
2.3: Speel-modus
2.4: Reset-modus
2.5: Game-over
2.6: Game-won

Elke toestand bezit zijn eigen methode. Deze methode voorziet al het noodzakelijke om de opdrachten voor die mode uit te voeren. Voor de beschrijving van deze methodes verwijzen we naar de documentatie. Hieronder wordt een korte toelichting gegeven over de belangrijkste zaken.

2.1 start_screen

Het startscherm wordt afgebeeld met methodes uit de Pygame module. Dit wordt verwezenlijkt met image.load(file_path), blit(image, position) en flip(). Daarnaast doet deze methode ook een check of het spel gesloten of gestart moet worden. Toets “X” moet worden ingedrukt om het spel te starten(check_x_event()) en de close button, van het speelvenster, om het spel te sluiten(check_quit_events()).

2.2 ready_screen

Het ready scherm beeldt een nieuw spel af op het display m.b.v. de draw methodes uit de klasse Map. Daarna speelt er een intro muziekje en wordt de tekst "READY" weergegeven. Wanneer dit muziekje afgespeeld is, checkt de methode of het scherm ondertussen werd gesloten met check_quit_events().

2.3 play_screen

Deze methode is verantwoordelijk voor het speelscherm. Dit scherm is in tegenstelling tot ready_ niet meer statisch is. Zowel Pacman als de spoken worden afgebeeld en bewegen. Deze methode zal de Ghosts verzoeken om te kiezen welke bewegingsmodi ze zullen uitvoeren ook worden er enkele dingen gecontroleerd . Deze controles zijn onder andere of een Ghost Pacman opeet, Pacman die Candies opeet...

2.4 reset_screen

De Ghosts en Pacman worden terug op hun startposities geplaatst en hun default waardes worden weer ingeladen.

2.5 gameover_screen

Eerst wordt er een dead-animatie afgespeeld waarin pacman langzaam verdwijnt. Daarna wordt de lege map flikkerend afgebeeld met de highscore. Er wordt opnieuw gecontroleerd of de "X" toets is ingedrukt om het spel te herstarten. (Bijlage 5 – Afbeelding A)

2.6 game_won

Indien alle Candies zijn opgegeten wordt het level beëindigd en kan het volgende worden gestart. (Bijlage 5 – Afbeelding B)

3. Maze

Idealiter is het speelveld van Pacman makkelijk te veranderen, zodat indien gewenst er ook een ander doolhof gebruikt kan worden tijdens een andere speelsessie. Om dit te verwezenlijken wordt er een tekstbestand ingelezen. Hierin moeten echter wel dezelfde lettertekens worden gebruikt als in het oorspronkelijke bestand. Elk letterteken wordt geassocieerd met een afbeelding. Met behulp van een dictionary worden deze relaties gelegd en zo kan men om het even welke map bouwen door de juiste lettertekens te gebruiken als bouwstenen.

In het spel zal de constructor van de Maze klasse worden aangeroepen worden bij het opstarten van de Game. Er werd gekozen voor een set van ‘tiles’. Een ‘tileset’ is de verzameling van afbeeldingen nodig om een map aan te maken. De map houdt ook een set met “Candy” objecten bij, dit zijn de bolletjes die pacman op eet. Voor de gebruikte ‘tilesets’ zie [15] en [16].

Bij constructie worden volgende zaken bijgehouden in het Map object:

- Aantal rijen/kolommen
- Game object waarop de map van toepassing is
- Hoogte/breedte per tegel
- Tegelgrootte
- Map informatie uit tekstbestand

Daarnaast zijn er ook draw methodes voor alles wat op het scherm moet worden afgebeeld:

dr **aw_tekst** : een methode dat een string en coördinaat meekrijgt, waarmee op een bepaalde positie tekst kan afgebeeld worden.

3.1 draw_tekst

Een methode dat een string en coördinaat meekrijgt, waarmee op een bepaalde positie tekst kan afgebeeld worden.

3.2 draw_score

Beeldt de huidige score af linksboven op het scherm.

3.3 draw_grid

Voornamelijk voor debugging, beeldt roosterlijnen af die de individuele tegels representeren zodat de positie van elk object goed zichtbaar wordt.

3.4 draw_pacmandeathani

Zorgt voor de animatie wanneer pacman wordt opgegeten door een Ghost.

3.5 init_items

Deze methode maakt alle objecten aan (Gates, Candies, SuperCandies, Ghosts en Wall).

3.6 init_tiles

Vult de tile array "self.__tiles" in met bijhorende tegelafbeeldingen.

4. Candy en Fruit

Er zijn 3 soorten “voedsel” voor Pacman, voor de gewone bolletjes kozen we de klassennaam “Candy”, dit is eveneens de superklasse van SuperCandy. Elke Candy heeft een coördinaat en een score. Deze wordt aan pacman toegekend wanneer hij deze op eet.

SuperCandy is een speciale variant van het gewone Candy-object. Indien Pacman dit soort voedel opeet worden de Ghosts eetbaar.

Als laatste soort voedsel bestaat er Fruit, dit is een object dat slechts op enkele momenten wordt weergegeven. Indien zo een object wordt opgegeten krijgt de speler, naar gelang welk stuk fruit het is, meer punten dan bij een normaal Candy of SuperCandy. Voor de puntenverdeling verwijzen we naar Bijlage 5.

Bij constructie van een Candy-object wordt het game display meegegeven waarop de Candy wordt afgebeeld. Ook wordt de coördinaat waar de candy terecht moet komen en de afbeelding die de candy voorstelt meegeven.

Verder is er een draw methode om de candy te tekenen op zijn coördinaat. Ook is de vergelijkingsoperator overschreven die nagaat of twee Candies dezelfde zijn. Deze controle op gelijkheid gebeurt op basis van hun coördinaat (dezelfde coördinaat betekent dezelfde candy).

Tenslotte zijn er nog getters voor de coördinaat en de score van de candy. Voor al deze methodes verwijzen we naar de documentatie[13].

5. FruitSelector

Om te weten wanneer net welke Fruit moet worden getoond, is er extra logica nodig, die zinloos is om aan elke Fruit op zich te geven, dit zou dan ook extra logica aan de Game moeten geven. We hebben besloten om deze logica te verschuiven naar de klasse FruitSelector, die alle Fruits zal beheren. Bij aanvang van het level zal de FruitSelector twee Fruits aanmaken die doorheen het level tijdelijk zullen worden getoond; als Pacman een bepaald aantal Candies heeft opgegeten. M.b.v de fruit_active() methode die FruitSelector aanbiedt, kan Pacman te weten komen wanneer er een Fruit actief is. Als Pacman het getoonde Fruit op eet, weet de FruitSelector net welk fruit er werd opgegeten en zal deze bonuspunten aan Pacman toekennen. De FruitSelector zal ook in de rechteronderhoek van het scherm weergeven welke Fruits Pacman al heeft opgegeten.

6. Coordinate

Om alles betreffende coördinaten bij te houden is er een klasse Coordinate, deze houdt x en y coördinaten bij die de tegels van de Maze representeren. Vrijwel alle 'visuele' klassen (vb. Candies, Pacman...) hebben nood aan een coördinatenstelsel voor enerzijds hun positie bij te houden en anderzijds aan botsingsdetectie te kunnen doen (zowel tussen een Character en een muur als Pacman en een Ghost). Elke Coordinate weet ook of hij een muur is of niet. De belangrijkste functionaliteit van deze klasse is de logica aanbieden voor deze andere objecten betreffende positie.

Bij constructie houdt de coördinaat dus twee getallen bij die de x en y coördinaten voorstellen, daarnaast weet elke coördinaat of hij een muur is of niet m.b.v de boolean is_wall.

Er zijn ook getters voor deze variabele, en ook voor de x en y coördinaten, zowel afzonderlijk als samen in een tuple. Deze kunnen ook worden omgezet naar de pixelwaarde, wat de Maze helpt om uiteindelijk alles te tekenen. De andere methodes zijn opnieuw te vinden in de documentatie.

7. Gate

Dit is een klasse die werd geschreven voor de tunnels/poorten waar Pacman en de Ghosts kunnen doorgaan, en aan de andere kant kan uitkomen. Een poort heeft een coördinaat en deze klasse bevat een methode om dat op te vragen. Er is ook een methode met als parameter de coördinaat van één poort en heeft als teruggeefwaarde de coördinaat van de andere poort[13]. Er zijn er altijd twee die ‘verbonden’ zijn.

In het spel zit maar één paar van deze poorten. Deze klasse werd gemaakt om de poorten uit te breiden naar bijvoorbeeld andere plaatsen of om meer dan één paar poorten aan te maken, dit kan worden gedaan door de Maze een ander tekstbestand te laten lezen.

8. Character

Nu Maze en Candy er zijn, moeten Pacman en de spoken nog gemaakt worden. Er werd ingezien dat beiden veel gemeenschappelijke kenmerken hebben: ze bewegen op dezelfde manier (vloeiende bewegingen door de Maze), worden op dezelfde manier getekend, doen allebei aan botsingsdetectie met muren...

Om de gemeenschappelijke kenmerken niet twee keer te moeten schrijven, wordt er een abstracte klasse Character gemaakt, dit word in Python verwezenlijkt door af te leiden van de klasse ABC[12]. Een abstracte klasse heeft tenminste één abstracte methode, in dit geval is het de methode move() omdat deze methode in alle afgeleide klassen anders geïmplementeerd wordt. De rest van de methodes van Character zijn bijna identiek voor Pacman als voor Ghost.

Characters bewegen op een manier zodat er een “smooth transition” is van de oude tegel naar de nieuwe met de methode move_between_tiles, dit is ook de methode die de coördinaat van Character zal updaten.

Pas als een Character perfect op de tegel staat wordt gecheckt of hij wel naar de volgende tegel kan bewegen. Dit wordt gedaan door de variabele is_wall van de volgende Coordinate af te tasten. Indien deze op true staat, zal hij stil staan tot wanneer zijn Direction wordt veranderd (wordt

verwezenlijkt door de move() methode, die overgelaten is aan de subklassen). Pas dan wordt de variabele moving_between_tiles gezet en zal de Character naar de volgende tegel bewegen.

Hieronder bevinden zich voorbeelden van de gemeenschappelijke methodes. Voor alle methodes in deze klassen wordt verwezen naar de documentatie[13].

8.1 move_between_tiles

Characters verplaatsen zich vloeiend van tegel naar tegel over enkele frames, niet onmiddellijk, tijdens deze overgang wordt deze methode telkens aangeroepen, die ervoor zorgt dat een Character enkele pixels (gedefinieerd door de variabele speed) verder wordt getekend. Pas op het einde, als een Character volledig op de volgende tegel staat, wordt zijn coördinaat geüpdatet.

8.2 set_on_opposite_side

Deze methode implementeert de beweging die gebeurt wanneer Pacman of een Ghost door een Gate gaan. Wanneer dit gebeurt wordt de coördinaat van Character vervangen naar de coördinaat aan de overkant van het spel.

8.3 calculate_new_coord

Deze methode berekent de volgende coördinaat gebaseerd op de huidige bewegingsrichting en houdt ook rekening met het geval waar één van de gates gebruikt wordt met behulp van de jump variabele.

8.4 reset_character

Deze methode reset de coördinaat van de Character naar zijn startcoördinaat, hierbij is het belangrijk dat een diepe kopie gemaakt wordt van start_coord.
Stel dat dit niet wordt gedaan, dan zal er een referentie naar start_coord gebruikt worden en zullen aanpassingen op de huidige coördinaat ook gebeuren op de start_coord.

8.5 get_coord en get_direction

Beide geven een diepe kopie terug, omdat we niet willen dat de coord of direction door een ondiepe kopie veranderd kunnen worden.

8.6 set_speed

Een setter voor de snelheid van de Character.

9. Direction

De klasse Direction is een enum, deze klasse bepaalt de mogelijke richtingen waar de characters in kunnen bewegen. In dit geval zijn dus de richtingen links, rechts, naar boven, naar onderen en stilstaan gedefinieerd. Iedere Direction is samengesteld uit een x en y waarde, die deze richting definieerd. Als een Character naar rechts wil bewegen moet de x waarde van zijn coördinaat met 1 vermeerderd worden, daardoor wordt Direction.RIGHT gedefinieerd als (1, 0) , analoog voor de andere richtingen.

9.1 get_reverse

Kijkt in een dictionary waarin alle richtingen op hun tegengestelde afgebeeld zijn en geeft de omgekeerde richting terug.
Bijvoorbeeld: Direction.LEFT geeft Direction.RIGHT terug.

9.2 get_letter

Geeft de eerste letter van de richting terug, deze methode wordt gebruikt voor de image van Pacman dynamisch in een richting te laten bewegen.
Bijvoorbeeld: Direction.RIGHT geeft dan “r” terug.

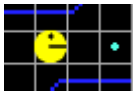
10. Pacman

De Pacman klasse erft over van de abstracte klasse Character en implementeert de methode move(). Deze klasse moet, naast zichzelf tekenen op het scherm en zijn coördinaten onthouden wat al door Character wordt verwezenlijkt, veel meer functionaliteiten hebben. Zo moet Pacman extra parameters bijhouden over score, aantal levens, streak, aantal Candies die nog gegeten kunnen worden en of het een SuperCandy heeft opgegeten.

Pacman bestaat uit verschillende foto’s bijvoorbeeld mondje open en mondje dicht.

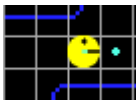
Het grote verschil met alle andere Characters is dat Pacman wordt gecontroleerd door user-input, want dit is het object dat de speler bestuurt. De user-input is wat de Direction van Pacman zal bepalen.

Bij Pacman is move_between_tiles nog wat aangepast: als Pacman beweegt van de ene tile naar de andere, negeert hij in die korte tijd alle input en beweegt hij naar de volgende tile (tenzij hij zich wil omkeren). Enkel als Pacman perfect op een tegel staat, bekijkt hij (eventueel eerdere) ingegeven input, en kijkt hij of hij in deze richting kan bewegen, anders stockeert hij deze richting tijdelijk in de variabele change_direction tot hij het wel kan. In figuren 1 tot 3 wordt move_between_tiles wat grafischer weergegeven (maar wordt voor eenvoud aangenomen dat Pacman 'alle' input negeert, terwijl hij nog steeds checkt op input om zich om te keren)



Figuur 1: Checkt input,...

Figuur 2: Moving_between_tiles, "negeert" input



Figuur 3: Checkt input,...

Voor meer informatie wordt opnieuw verwezen naar de documentatie.

11. Ghost

Deze klasse implementeert ook de abstracte/superklasse Character en kan dus ook alle methodes ervan gebruiken. Om goed werkende Ghosts te krijgen zijn er natuurlijk nog enkele andere methodes nodig. De belangrijkste methodes worden hieronder opgesomd:

11.1 move_selector

Deze methode kijkt welke soort move uitgevoerd moet worden, dit wordt verder uitgelegd in de paragraaf "Activiteitdiagram van move_selector".

11.2 move_to_start

Dit is een belangrijke methode voor als een Ghost opgegeten is door Pacman, het zorgt er namelijk voor dat de Ghosts terug naar het Ghosthuisje bewegen. Dit gebeurt aan een drievoud van de normale snelheid, zodat de opgegeten Ghost zo snel mogelijk weer in het spel terecht komt.

11.3 update_target_tile en update_target_tile_scatter

Beide methodes berekenen een nieuwe target-tile voor respectievelijk de move methode en de scatter methode. In de paragraaf “Ghost bewegingsmodi” wordt er dieper op ingegaan hoe ze deze tiles berekenen.

move_selector

Beide geven een diepe kopie terug, omdat we niet willen dat de coord of direction door een ondiepe kopie veranderd kunnen worden.

- **move_selector:**
  Deze methode kijkt welke soort move uitgevoerd moet worden, dit wordt verder uitgelegd in de paragraaf "Activiteitdiagram van move_selector".

- **move_to_start:**
  Dit is een belangrijke methode voor als een Ghost opgegeten is door Pacman, het zorgt er namelijk voor dat de Ghosts terug naar het Ghosthuisje bewegen. Dit gebeurt aan een drievoud van de normale snelheid, zodat de opgegeten Ghost zo snel mogelijk weer in het spel terecht komt.

- **update_target_tile en update_target_tile_scatter:**
  Beide methodes berekenen een nieuwe target-tile voor respectievelijk de move methode en de scatter methode. In de paragraaf “Ghost bewegingsmodi” wordt er dieper op ingegaan hoe ze deze tiles berekenen.

- **move(), frightened() en scatter():**
  Dit zijn de drie verschillende bewegingsmodi:

- De move() methode wordt uitgevoerd als de Ghosts zich in chase-modus bevinden.
- De frightened() methode wordt uitgevoerd als Pacman een SuperCandy heeft opgegeten.
- De scatter() methode wordt uitgevoerd als er geen van de bovenstaande methodes worden uitgevoerd.

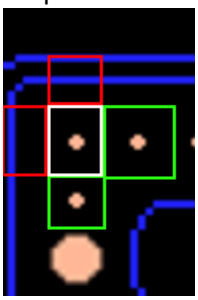
Meer info over de afwisseling van deze modi volgt in een volgende paragraaf.

12. Astar – Algoritme

Het Astar object geeft eigenlijk hersenen aan de Ghosts, dit object implementeert namelijk het A*-algoritme. Onze implementatie is gebaseerd op de implementatie van referentie [9], maar is verbeterd en geoptimaliseerd voor de toepassingen die nodig zijn in het spel.

Deze klasse heeft veel hulp methodes, die nodig zijn voor alle berekeningen uit te kunnen voeren of voor bepaalde datastructuren te maken, zoals bijvoorbeeld een graaf van de Maze. Hieronder zullen de belangrijkste methodes die de Ghosts nodig hebben, verder worden uitgelegd:

- Een belangrijke methode van het Astar object is de make_graph() methode, deze methode maakt namelijk een soort van graaf aan van het spelbord. De graaf wordt opgeslagen als een dictionary met als key een bewandelbare coördinaat (in tuple vorm) en als value een array van tuples. Deze tuples zijn opgebouwd uit een buur waarnaar gelopen kan worden en de richting dat de buur staat ten opzichte van de key.
  Bijvoorbeeld: De witte tile heeft coördinaat (1,1). De groene vierkanten in Figuur 4 stellen tiles voor waarnaar bewogen kan worden, deze liggen rechts (RIGHT) en onder (down) de witte tile. In de graaf wordt dit dan voorgesteld als: {(1,1) : [('D', (1, 2)), ('R', (2, 1))]}, met R(IGHT) en D(OWN) de richting ten opzichte van de key tile.



Figuur 4: Visueel voorbeeld voor opstelling van de graaf

- De heuristic(start, goal) methode is de heuristische functie, deze methode heeft 2 parameters, namelijk 2 tiles. Hiervan berekent het de kleinst mogelijke afstand tussen beide tiles, er wordt ook rekening gehouden met de Gates. Dus als er een afstand korter is als er een gate genomen wordt, zal die waarde teruggeven worden.

- get_direction(start, goal) geeft de eerste en dus de beste richting terug die een Ghost moet nemen om volgens het kortste pad van de start tile naar de goal tile te lopen.
- De belangrijkste methode van deze klasse is uiteraard de find_path(start, goal) methode, deze methode wordt in de normale modus van het spel bij iedere intersectie of bocht opgeroepen, zie groene tegels Figuur 11.

Deze methode werkt volgens het A*-algoritme. In onze implementatie werken we met een heap. Dit kan gezien worden als een “soort stack” met als eerste element het pad dat momenteel de kleinste kost heeft. Hier wordt dit de heap invariant genoemd, de heap wordt dus altijd gesorteerd met de kleinst kost eerst en de grootste kost laatst.
In code voorbeeld 1 wordt de code mooi getoond, aan de hand van deze afbeelding zal het algoritme verder uitgelegd worden. Als eerste stap worden volgende variabelen als 1 geheel in de heap gestoken:
(Cel heeft dezelfde betekenis als tile)

- De afstand tussen de begin- en doeltegel
- De kost van het pad (=0, is een leeg pad)
- Een pad van de start cel tot de huidige cel
- De huidige cel, dit is de cel die je bekomt als je van de start cel het pad (van hierboven) volgt. Dit is hier dus ook gelijk aan de start cel omdat het nog een leeg pad is.

Daarna zullen onderstaande stappen steeds herhaald worden totdat de heap leeg is of tot er een pad gevonden is. Als de heap leeg is en de laatste huidige cel is niet de doeltegel, dan wil dat zeggen dat er geen pad is die de start cel kan verbinden met de doel cel.

1. Het momenteel meest belovende pad wordt van de heap gehaald (kleinste kost) en wordt in vier variabelen gesplitst:
    a. De variabele total_cost is de kost van het pad (zie b) opgeteld met de afstand tussen de huidige cel (current_cell) en de doel cel
    b. De variabele cost is de kost van het berekende pad (zie c), dit is hier eveneens de lengte van het pad, omdat iedere stap een gewicht van 1 bij de totale kost optelt.
    c. De string path is het pad dat je van de start cel naar de huidige cel brengt (current_cell)
    d. Als laatste is er de variabele current_cell, dit is de cel waarin je eindigt als je het pad van hierboven uitvoert met als begin de start cel. In de verdere uitleg zal deze variabele voornamelijk als huidige cel vernoemd worden.

2. Er wordt gekeken of de huidige cel gelijk is aan de doel cel, als dit het geval is wordt het pad direct terug gegeven.

3. Hierna wordt er gekeken of de huidige cel al in de verzameling van bezochte cellen zit:
    a. Indien het nog niet in de verzameling zit zal het eraan toegevoegd worden.
    b. Als het er wel al inzit wordt stap 1 weer uitgevoerd, de reden hierachter is zeer eenvoudig. Als een cel al in bezochte cellen verzameling zit wil dit zeggen dat er al een pad (A) geweest is, die dezelfde cel als eind cel had. Doordat we met een heap werken die gesorteerd is, wil dit zeggen dat pad A dus al een beter pad was dan het pad waarvoor nu gekeken wordt. Daarom kan er besloten worden dat er nooit een beter pad, later in het algoritme kan ontdekt worden die een cel bevat dat ervoor al eens bekeken werd .
4. Na 3.a worden alle buren, van de huidige cel, overlopen en worden er nieuwe items naar de heap gepusht, namelijk:
    a. Het nieuwe pad, dit bestaat uit het oude pad + de richting waarin deze buur staat ten opzichte van de huidige cel.

b. De totale kost = kost van het nieuwe pad + afstand van deze buur tot de doel cel.
c. De kost van het pad.
d. De huidige cel, dit wordt dan de desbetreffende buur.

Deze stappen worden uitgevoerd tot de huidige cel gelijk is aan de doel cel, of tot de heap leeg is. Het A*-algoritme is een “Best First Search”- algoritme, dit wil zeggen dat de bovenstaande stappen steeds op het huidige beste pad uitgevoerd worden. Dit wordt hier dus verwezenlijkt doordat er een heap gebruikt wordt, die alles mooi gesorteerd houdt. In Bijlage 4 wordt het algoritme nog eens visueel getoond a.d.h.v. een activiteitendiagram.

Code voorbeeld 1: Het A-algoritme*

```
heap = []
start, goal = start_coord.get_coord_tuple(),
              goal_coord.get_coord_tuple()
heappush(heap, (0 + self.heuristic(start, goal), 0, "", start))

visited = set()

while heap:
    total_cost, cost, path, current_cell = heappop(heap)
    if current_cell == goal:
        return path if len(path) > 0 else "B"
    if current_cell not in visited:
        visited.add(current_cell)
        try:
            for direction, neighbour in self.__graph[current_cell]:
                heappush(heap, (cost + self.heuristic(neighbour, goal),
                                cost + 1, path + direction, neighbour))

        except:
            pass
return "NO PATH"
```

13. MusicPlayer

Wat is een spel zonder geluid? Alle zintuigen van de mens moeten natuurlijk aangesproken worden in het spel. Hiervoor werd de klasse MusicPlayer in het leven geroepen.

Bij constructie kan een bepaalde pad meegegeven worden, dit pad dient om aan te duiden waar de geluidsbestanden zich moeten bevinden (of in een onderliggende folder). Bevindt het bestand zich daar niet, dan kan het niet worden afgespeeld (met deze MusicPlayer toch). Er zijn drie methodes beschikbaar. De eerste dient om eender welk geluid af te spelen, de tweede voor de typische achtergrondmuziek af te spelen en de derde om de achtergrondmuziek te stoppen. De eerste methode speelt een geluidsbestand eenmalig af. In tegenstelling tot de eerste methode, wordt de tweede methode constant aangeroepen (door Game in de play_screen loop) omdat de achtergrondmuziek tijdens het spel steeds moet worden afgespeeld. De geluiden die tijdens het spel gebruikt worden, komen van ClassicGaming[14].

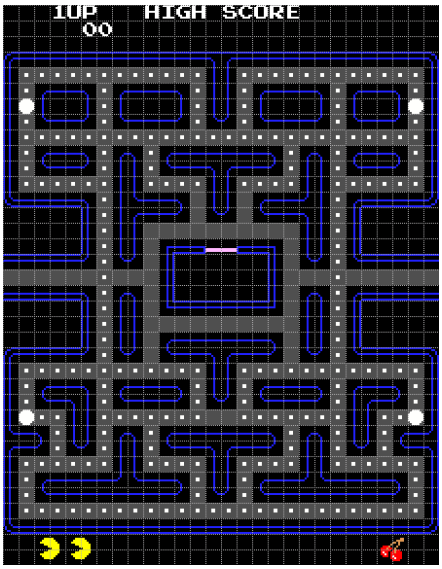
14. Verduidelijking functionaliteiten

14.1

Doolhof logica

Het volledige veld heeft een breedte van 448 en een hoogte van 576 pixels. Dit veld is onderverdeeld in tegels die elk 16 op 16 pixels zijn. Hierdoor worden er 28 kolommen en 36 rijen verkregen.

De actoren (Pacman en de verschillende Ghosts) bewegen enkel in de 'legale ruimte' ingekleurd in het grijs. De locatie van een actor in het doolhof wordt bepaald door middel van het vak (wat dan overeenstemt met een coördinaat) waarin hij zich bevindt. Hierop is de botsingsdetectie gebaseerd. Maar bij het bewegen van tegel naar tegel zal een actor slechts één tegel (één coördinaat) in beslag nemen ook al bevindt hij zich grafisch in 2 tegels. De huidige tegel waarin een Character zich bevindt wordt pas geüpdatet als het volledig in de volgende tegel staat.

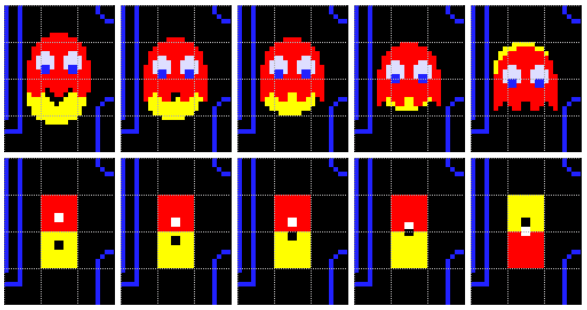


Figuur 5: doolhof onderverdeling[1]

Dit botsingsmechanisme op basis van tegels is niet het meest nauwkeurige. Beter zou zijn om te kijken naar de pixels die het spookje en pacman in beslag nemen; vanaf één van de vele pixels kruisen, wil het zeggen dat ze zijn gebotst.

Er werd gekozen voor botsingsdetectie op basis van tegels aangezien het originele pacman spel hierook op is gebasseerd, en dit ook veel eenvoudiger te implementeren is.

Hierdoor krijg je een interessante bug/easteregg(verborgen truc).
Bij het bewegen van pacman kan je, bij exacte timing, door het een spookje bewegen zonder dat je hiervoor dood gaat. Dit is doordat de coordinaten pas worden aangepast als je in de volgende tegel staat. (zie figuur 6) Dit is ook zo in het originele spel.

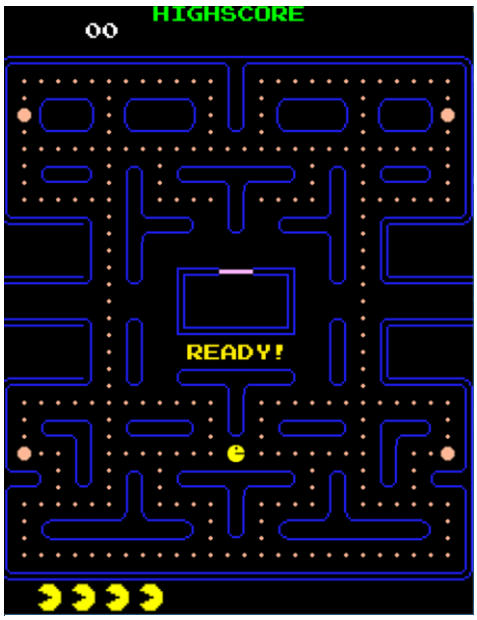


Figuur 6: botsingsmechanisme bug

Aanvang

Pacman en de 4 spookjes worden bij aanvang telkens op een vaste locatie in het doolhof gezet. De map wordt telkens geïnitialiseerd met dezelfde hoeveelheid Candy (kleine oranje bolletjes) en SuperCandy (grote oranje bolletjes). Al deze gegevens worden uit het tekstbestand gehaald dat eerder bij Maze werd aangekaart.

Bij aanvang heeft Pacman nog 3 levens over. Hij is namelijk al in zijn 4de leven actief.



Figuur 7: scherm bij aanvang van spelen

Figuur 8: scherm gedurende het spel

14.2 Gameplay

Pacman kan vrij bewegen in het doolhof, hij moet echter wel oppassen van de spookjes die hem kunnen opeten. Doorheen het spel kunnen de Ghosts in verschillende modi terechtkomen (zie paragraaf Ghost behaviour). Denk maar aan vb. Pacman die een SuperCandy op eet, waardoor de Ghosts tijdelijk in frightened mode terecht komen. Deze implementaties kunnen het spel een andere wending geven, wat de speelstrategie zeker zal beïnvloeden. Men moet de juiste keuzes maken om zolang mogelijk te kunnen overleven.

Afhankelijk van het aantal Candies die Pacman op eet wordt er Fruit getoond door de FruitSelector. Dit gebeurt gedurende elk level 2 keer. De speler kan kiezen om hiervoor uit zijn weg te gaan en extra bonuspunten te verzamelen, of hij kan deze negeren, na verloop van tijd haalt de FruitSelector het fruit dan weer weg.

Soort	**Level**
Cherry	1
Strawberry	1
Orange	2
Apple	2

In tabel 1 wordt er een overzicht gegeven van welk Fruit op welke levels voor komt. Door gebrek aan afbeeldingen en doordat velen niet voorbij de eerste levels raken, zijn de soorten Fruit gelimiteerd.

Green Melon	3+

Zoals reeds vermeld is het verschijnen en verdwijnen van een stuk fruit afhankelijk van het aantal Candies die in de Maze over zijn, dit wordt weergegeven in tabel 2.

Candy over in doolhof:	**Minimum**	**Maximum**
1ste stuk	180	140
2de stuk	60	100

Tabel 1:stukken fruit spawnen afhankelijk van de hoeveelheid candy in het doolhof over

Tabel 2:afhankelijk van het level een ander stuk fruit

14.3 Doel

Het speldoel is om een zo hoog mogelijke score te halen tot als alle levens op zijn. Indien de speler de hoogst behaalde score heeft, wordt dit bij elke volgende sessie zelfs getoond tijdens het spel zelf, tot wanneer deze score wordt verslagen door iemand anders. Zo kan men constant streven naar een betere highscore, een eigen persoonlijk record of iemand anders zijn score verslaan. In de volgende paragraaf wordt uitgelegd hoe je als speler score kan behalen. Het volledige spel uitspelen is onmogelijk aangezien de Game na elk level direct het volgende opstart. Hierdoor kan een speler met genoeg talent in theorie een oneindig grote score behalen.

Figuur 9: startscherm waarop de 5 beste scores worden afgebeeld

14.4

14.5 Score berekening

Score kan de speler verdienen op verschillende manieren. De meest voor de hand liggende is het verzamelen van de normale Candies die talrijk verspreid liggen over het doolhof. De gevaarlijkste manier maar ook degene waarmee je potentieel het meeste punten kan verdienen is bij het nemen van de SuperCandies. Hierdoor komen de Ghosts in 'frightened mode' terecht (zie p13), waardoor de rollen worden omgedraaid en de speler de Ghosts nu kan gaan opeten. Een bijkomende manier om punten te verdienen is door het verzamelen van Fruit. Dit wordt na enkele levels cruciaal om een hoge score te behalen. In tabel 3 staat een overzicht van alle score bronnen.

Item	**Punten**
Pac-Dot (Candy)	10
Power Pellet (SuperCandy)	50
1e ghost	200
2e ghost	400
3e ghost	800
4e ghost	1600
Cherry	200
Strawberry	400
Orange	600
Apple	800
Green Melon	1200

Tabel 3:overzicht puntenverdeling

14.6 Ghost bewegingsmodi

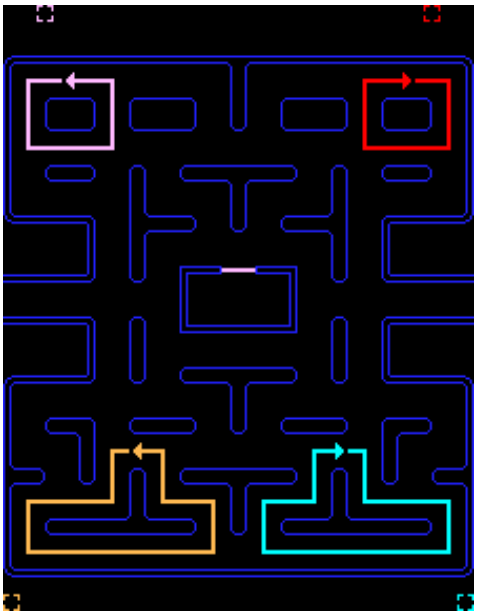
De Ghosts hebben drie verschillende bewegingsmodi, nl. de scatter-, frightened - en de chasemodus. Als we dit vertalen worden deze respectievelijk de verspreid -, bange - en de achtervolg modus. Deze modi zorgen er ook vaak voor dat elke Ghost zijn doeltegel wordt aangepast. Ieder spookje heeft zijn eigen persoonlijkheid en dus ook zijn eigen regels. In onderstaande paragrafen zullen alle modi per Ghost verder worden uitgelegd.

Scatter modus

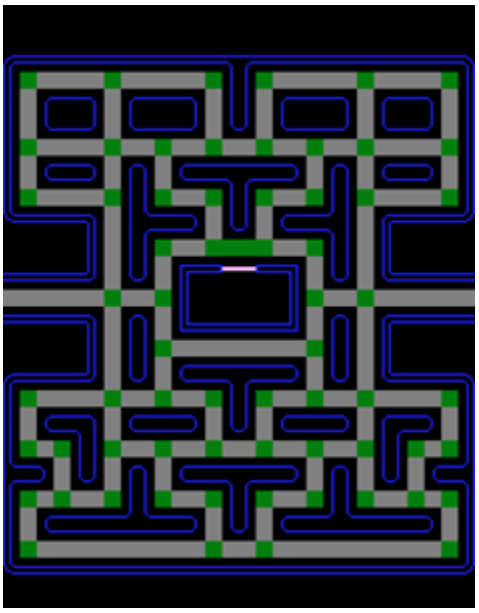
Iedere Ghost heeft in het spel zijn eigen hoek, in Figuur 10 worden deze hoeken afgebeeld. In de scatter modus zal de Ghost rondjes draaien in de buurt zijn eigen hoek omdat deze als doeltegel geselecteerd is. Deze modus heeft als doel om alle Ghosts te verspreiden, hierdoor wordt het voor de speler moeilijker omdat alle hoeken “bewaakt” worden. In deze implementatie van het spel duurt deze modus zeven seconden.

Frightened modus

Deze modus is voor alle Ghosts gelijk, hierin zijn ze kwetsbaar en kunnen ze opgegeten worden door Pacman. De Ghosts berekenen in deze modus op iedere intersectie, of bocht, zie Figuur 11, een willekeurig richting. Dit proces is slimmer gemaakt door ervoor te zorgen dat de Ghosts nooit de tegengestelde richting kiezen waarin ze (tot) nu gaan.
Bijvoorbeeld: indien een Ghost op intersectie 1 kiest voor de richting links, zal hij op de volgende richting niet kiezen voor richting rechts. Door dit principe toe te passen zullen de Ghosts dus nooit heen en weer bewegen. De duurtijd van deze modus hangt af van het aantal overblijvende Candies, hoe minder Candies er nog in het spel zitten, hoe minder lang deze modus duurt, wat de moeilijkheidsgraad voor de speler verhoogt.



Figuur 10: Scatter modus, elk spookje heeft een eigen "hoekje", referentie [2]



Figuur 11: Groene vakjes zijn intersecties of bochten [2], maar aangepast voor accurate weergave van eigen implementatie

Chase modus

Het grootste deel van de tijd zullen de spookjes zich echter in deze modus bevinden. Het principe van deze modus is dat de spookjes proberen om Pacman op te eten. Deze modus is ook voor iedere Ghost uniek, ze hebben elk een eigen doeltegel waarnaar ze proberen te bewegen. De filosofie die erachter zit is zeer eenvoudig, samen zouden de Ghosts in staat moeten zijn, door hun unieke

doeltegels, om Pacman te omsingelen. Hieronder wordt voor iedere Ghost hun eigen doeltegel verder uitgelegd, dit is gebaseerd op info die gevonden kan worden in referentie 2. Via het Astar object dat al eerder werd uitgelegd bepalen ze dan hun pad/richting.

De rode Ghost "Blinky":

Blinky's persoonlijkheid wordt in het Engels omschreven als "shadow", vertaald wordt dit "schaduw". Dit is een zeer goede beschrijving voor deze Ghost, want hij heeft als doeltegel de tegel waar Pacman zelf op staat. Dit zorgt er dus voor dat Blinky Pacman constant achterna zit, en dat de speler hem bijna niet kan kwijt raken. (Figuur 12)

De roze Ghost "Pinky":

Pinky's persoonlijkheid wordt in het Japans omschreven als "machibuse", vlotjes vertaald wordt dit iemand die een hinderlaag creëert. In de context van het spel betekend dit dat Pinky een hinderlaag probeert te maken voor Pacman. De doeltegel wordt dan ook als volgt bepaald: hij neemt de positie van Pacman en schuift deze vier tegels op, in de richting dat Pacman momenteel beweegt. Zo komt de tegel enkele plaatsen voor Pacman te liggen. Deze tegel wordt dan Pinky's doeltegel. (Figuur 13)

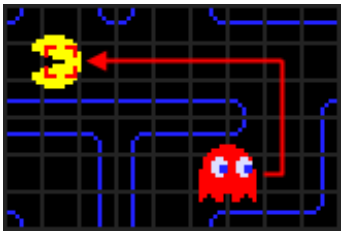
De blauwe Ghost "Inky":

Inky's persoonlijkheid wordt in het Engels omschreven als "whimsical", vertaald wordt dit "wispelturig". Deze toch wel vreemde omschrijving wijst erop dat het, voor de speler, zeer moeilijk te voorspellen is hoe Inky zal bewegen. Om Inky's doeltegel te berekenen zijn er twee dingen nodig, de coördinaat van Pacman en de coördinaat van de rode Ghost Blinky. Eerst en vooral wordt er een nieuwe coördinaat berekend die twee tegels voor Pacman ligt, daarna wordt er een vector van Blinky's positie naar de nieuwe coördinaat getrokken. Tenslotte wordt deze vector verdubbeld in lengte, de tegel waar deze vector eindigt is Inky's nieuwe doeltegel. (Figuur 14)

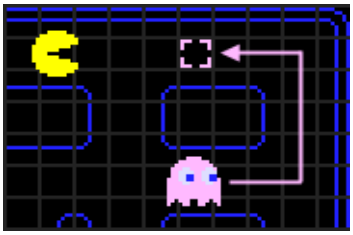
De oranje Ghost "Clyde"

Voor Clyde is er afgeweken van de doeltegel van in het originele spel, dit is zo gekozen om toch een speciale Ghost te creëren en een eigen twist aan het spel te geven. Er zijn twee mogelijke gevallen voor deze Ghost. Indien de afstand van Clyde tot Pacman kleiner is dan 10 tegels en groter dan 2 tegels, wordt de coördinaat van Clyde opgesplitst in een x en y coördinaat, bij zowel het x- als het y-gedeelte wordt er een verschillend willekeurig getal tussen -10 en +10 bij opgeteld. Clyde zijn doeltegel is dan gewoon de coördinaat met de nieuwe x en y waarde. Indien de afstand tussen Clyde en Pacman groter is dan 10 tegels wordt zijn doeltegel dezelfde tegel als waarop Pacman zich bevindt, dit is ook het geval als de afstand kleiner is dan 2 tegels. Het laatste fenomeen wordt Clyde's killer instinct genoemd. (Figuur 15)

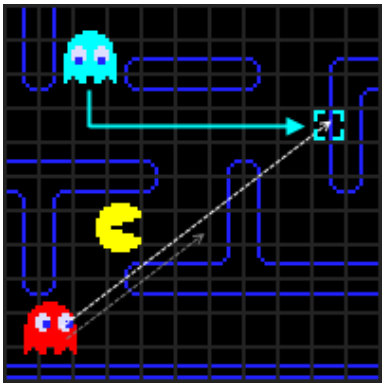
Indien er een doeltegel berekend wordt die buiten de Maze ligt, of zich op een niet bewandelbare tegel bevindt, dan zal het Astar-object van de Ghosts een nieuwe tegel berekenen die er zo dicht mogelijk bij ligt en waar ze wel naar kunnen bewegen. De chase modus duurt een 20-tal seconden.



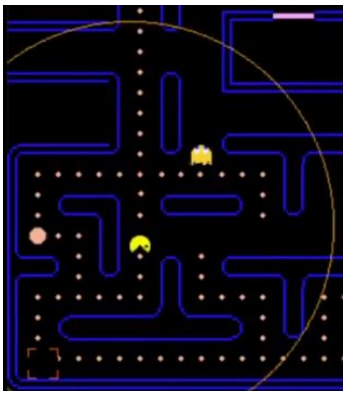
Figuur 12: Doeltegel van Blinky [2]

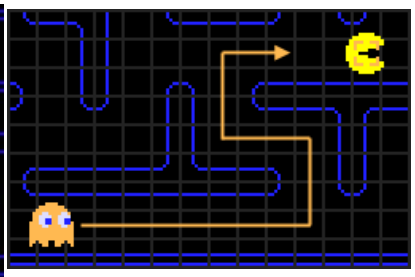


Figuur 13: Doeltegel van Pinky[2]



Figuur14: Doeltegel van Inky [2]





Figuur 15: Doeltegel van Clyde
Rechts [2], Links geen referentie

15. Relaties klassendiagram

Hoewel de klassen al goed werden beschreven, is het waarschijnlijk nog onduidelijk hoe sommige klassen met elkaar interageren. Wat dit kan verduidelijken is opnieuw het klassendiagram (bijlage 1) erbij nemen, waar alle klassen en hun onderlinge relaties worden weergegeven. In deze paragraaf worden de belangrijkste relaties wat aangekaart, voor de rest wordt verwezen naar het klassendiagram zelf.

In het klassendiagram staat de Game klasse centraal, die alle andere grote datastructuren zelf aan maakt en moet beheren voor vlotte communicatie. Game dient niet enkel als een lijm om alles samen te houden, maar moet ook bepalen of het spel moet worden gestopt of niet. Hiervoor is communicatie in beide richtingen vaak noodzakelijk, zo moet vb. Maze kunnen melden wanneer alle Candies opgegeten zijn, waarop Game zal dan reageren met de juiste gamemode te selecteren (nl. 'Gewonnen'). Zonder Game hebben andere klassen geen nut, en omgekeerd, zonder de andere klassen heeft Game geen functionaliteiten.

Een tweede belangrijke centrale klasse is Coordinate, die, alhoewel zeer compact, alle middelen aan biedt voor aan botsingsdetectie te kunnen doen. Zo zal vb. Pacman een candy opeten (of vb. een Ghost Pacman opeten) als hun coördinaat overeen komt. Het is dus vanzelfsprekend dat alle Characters en Candies nood hebben aan een eigen Coordinate (die al dan niet wordt aangepast in het geval van de Characters). Characters hebben ook nog een tweede Coordinate die hun startpositie bijhoudt. Gates hebben ook nood aan coördinaten, zodat als een Character over ze heen loopt, deze zijn coördinaat wordt aangepast naar de andere kant van de Maze.

Elke Ghost krijgt ook nog een eigen Astar object, dat eigenlijk de Ghost de belangrijste logica geeft: die om Pacman daadwerkelijk achterna te zitten. Zonder Astar zou de Ghost nooit kunnen achterhalen welke richting hij moet uitgaan, hij zou zelfs gewoon stil staan, wat we natuurlijk niet verwachtten als eindproduct. Daarom wordt aangenomen dat Astar daadwerkelijk noodzakelijk is voor Ghost. Natuurlijk anderzijds zal Astar nooit iets nuttigs doen zolang het niet aan een Ghost gekoppeld is.

16. Sequentiediagram

In het sequentiediagram (bijlage 2) wordt het initialisatieproces voorgesteld. Als de klasse Game geïnitialiseerd wordt, zal hij één of meer objecten van andere klassen initialiseren.

Eerst wordt de Pygame module[7] aangeroepen, dit is niet echt een object maar deze module is onmisbaar voor het spel. Het bevat namelijk het “canvas” om op te tekenen, modules om muziek af te spelen, modules om events te detecteren... Hierna worden alle bovenvermelde klassen geïnitialiseerd. Het Game object wordt teruggegeven en de methode run() wordt geactiveerd. Dit is de methode die in een oneindige lus zit, wat het spel draaiende houdt.

Er werd gekozen om een sequentiediagram te gebruiken om het initialisatieproces voor te stellen omdat men op deze manier kan zien welke objecten chronologisch worden aangemaakt en ook welke aanroepen er chronologisch gebeuren. Tijdens het spel zelf worden objecten en/-of methodes opgeroepen die af hangen van de speler en dus onmogelijk te voorspellen zijn.

Indien de afbeelding van het sequentiediagram niet goed zichtbaar is, kan gekeken worden naar de ingezoomde afbeeldingen in de map "Diagrammen".

17. Activiteitdiagram move_selector

De beweging van de verschillende Ghosts gebeurt op basis van één overkoepelende methode, net zoals de Game_Handler methode een delegerende functie heeft bij het schakelen tussen de verschillende game toestanden, werkt de move_selector op een analoge manier. De move selector schakelt hierbij uiteraard niet tussen gametoestanden maar wel tussen verschillende bewegingsmodi van de spoken.
Het bijhorende activiteitendiagram (Bijlage 4) maakt de werking wat duidelijker, op basis van een if-else structuur weet de move selector op welke manier het spook moet bewegen. Hierbij is het belangrijk om in te zien dat de eerste conditie kijkt of het spook terug naar start aan het bewegen is, dit mag op geen enkele manier overschreven worden door andere bewegingen (wat het geval zou kunnen zijn moest dit niet als eerste gecheckt worden). De bewegingen met hoogste prioriteit komen dus eerst in de move selector aan bod, als alles overlopen is en er geen sprake van speciale toestanden is, wordt de standaard move() methode gebruikt.

18. Besluit

Alles zal zelf geprogrammeerd worden met onze eigen ideeën en met een eigen twist. We baseren ons wel op verschillende bronnen en op het originele spel. Hierna komen alle bronnen/referenties die gebruikt zijn/zullen gebruikt worden, in de bijlages zit het klassendiagram, een sequentiediagram van de initialisatie van de Game, een activiteitendiagram van de belangrijke methode van de Ghosts (move_selector) en als laatste een activiteitendiagram van het A*-algoritme .

19. Referenties

[1] Jamey Pittman, “The Pac-Man Dossier”, geraadpleegd via https://www.gamasutra.com/view/feature/132330/the_pacman_dossier.php?page=9

[2] Chad Birch, “Understanding Pac-Man Ghost Behavior”, geraadpleegd via http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior

[3] Using Python, “Tilemaps”, geraadpleegd via http://usingpython.com/pygame-tilemaps/

[4] Sentdex, “Game Development in Python 3 With PyGame”, geraadpleegd via https://www.youtube.com/watch?v=ujOTNg17LjI&feature=youtu.be

[5] Paul Vincent Craven,” Teaching Python 3.6 with Games”, geraadpleegd via https://2017-craven-webinar.readthedocs.io/en/latest/

[6] David Reilly, “Pacman – 1.1”, geraadpleegd via http://www.pygame.org/project-Pacman-426-4585.html

[7] Pygame, “Pygame documentatie”, geraadpleegd via https://www.pygame.org/docs/

[8] Stackoverflow, “Get shortest path to a cell in a 2d array python”, geraadpleegd via https://stackoverflow.com/questions/47896461/get-shortest-path-to-a-cell-in-a-2d-array-python

[9] Bryukh, “Labyrinth Algorithms“, geraadpleegd via http://bryukh.com/labyrinth-algorithms/

[10] GeeksForGeeks, “Shortest path in a Binary Maze”, geraadpleegd via https://www.geeksforgeeks.org/shortest-path-in-a-binary-maze/

[11] Stackoverflow, “Solving a maze using recursion in python”, geraadpleegd via https://stackoverflow.com/questions/22702097/solving-a-maze-using-recursion-in-python

[12]Python 3.* doc, “Abstract base classes”,geraadpleegd via https://docs.python.org/3/library/abc.html

[13] Documentatie, “index.html’, geraadpleegd via http://users.ugent.be/~adlys/team-9/Documentatie/html/ of op Github Team-9/Documentatie

[14] ClassicGaming, "Pac-Man Sounds", geraadpleegd via http://www.classicgaming.cc/classics/pac-man/sounds

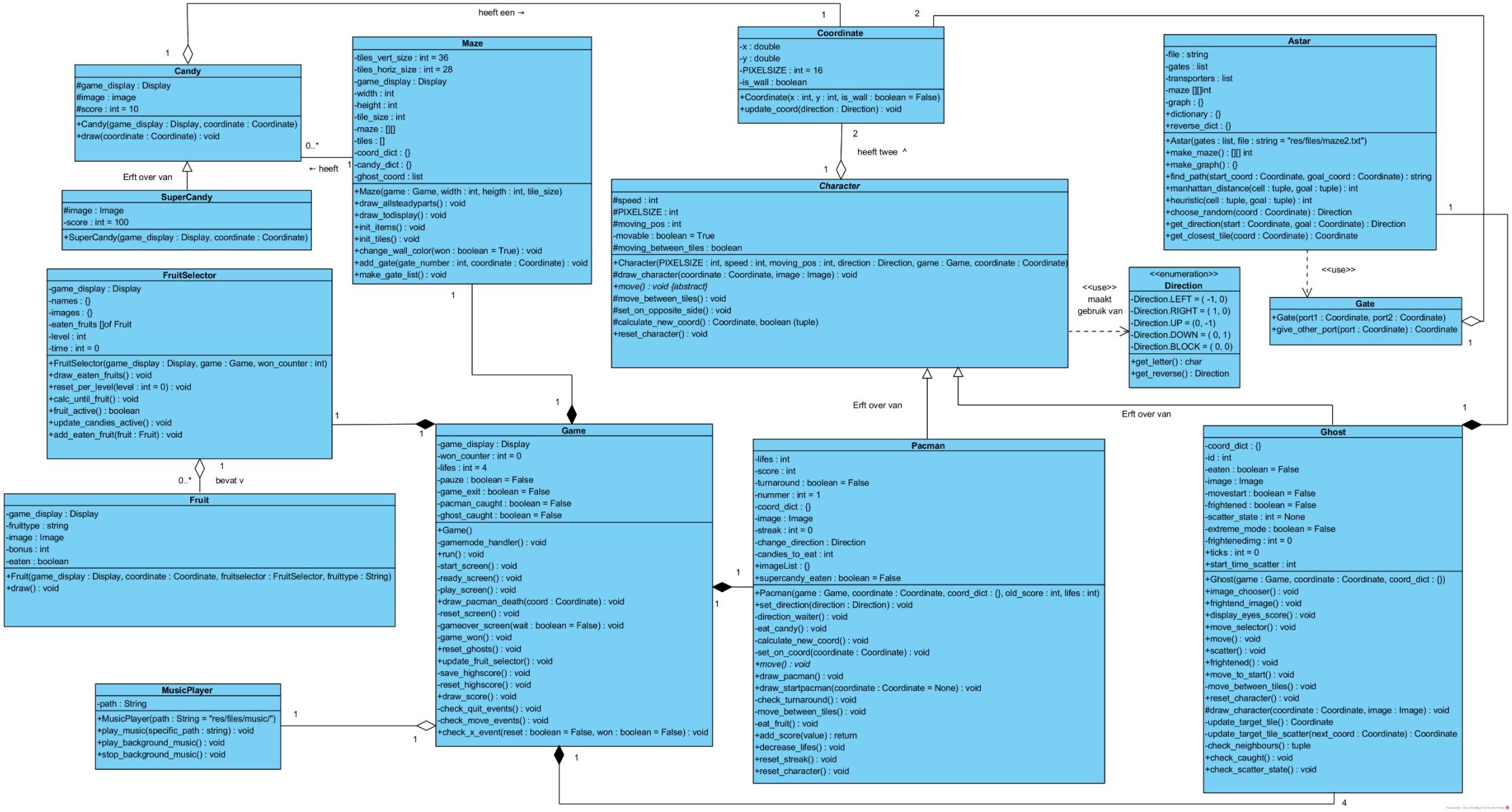
[15] Tileset deel 1, “Pacman tileset”, geraadpleegd via https://pixshark.com/ms-pacman-pixel.htm.mathworks.com/matlabcentral/answers/167389-combining-6-images-without-discoloration

[16] Tileset deel 2, “Combining 6 images without discoloration”, geraadpleegd via https://www.google.be/search?q=tileset+pacman&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjMnPjgqZnbAhUFU1AKHSNiCqwQ_AUICigB&biw=1778&bih=838#imgrc=7ZBkPmGpZJyNmM:

1. Bijlage 1

1.1 Klassendiagram

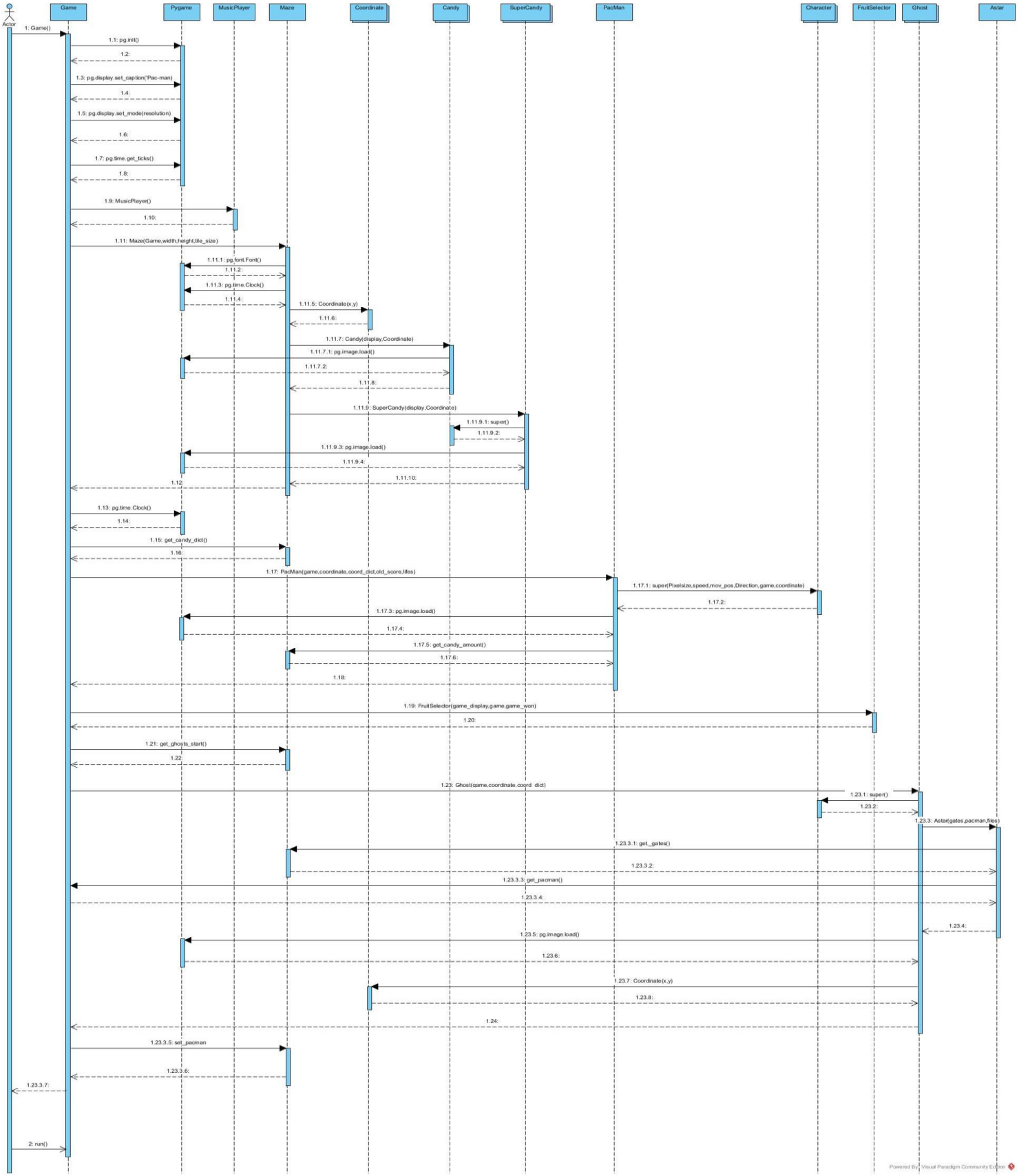
(Digitale versie zie Github)



20. Bijlage 2

20.1 Sequentiediagram

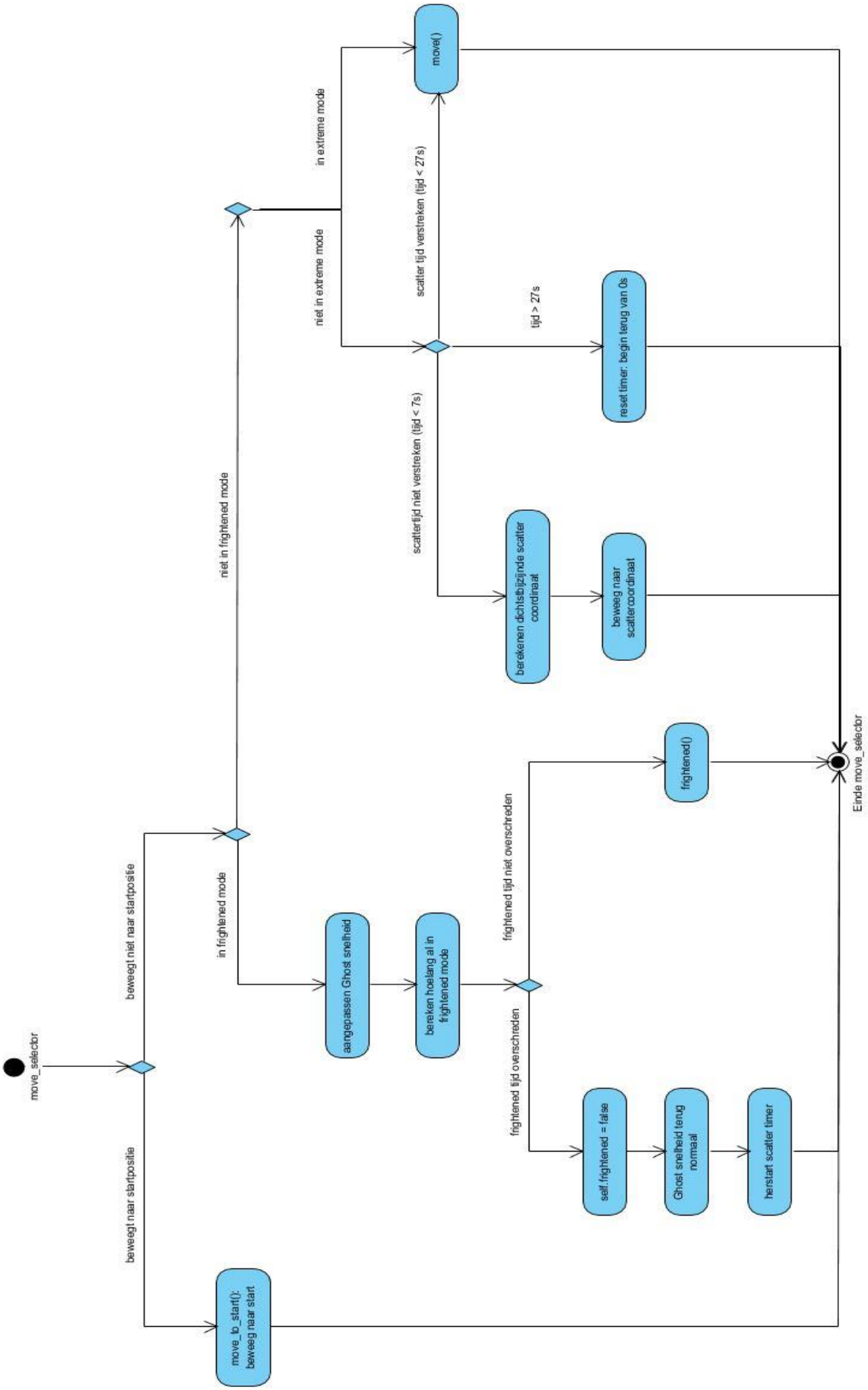
(Digitale versie zie Github)



21. Bijlage 3:

21.1 Activiteitendiagram voor de move_selector() methode

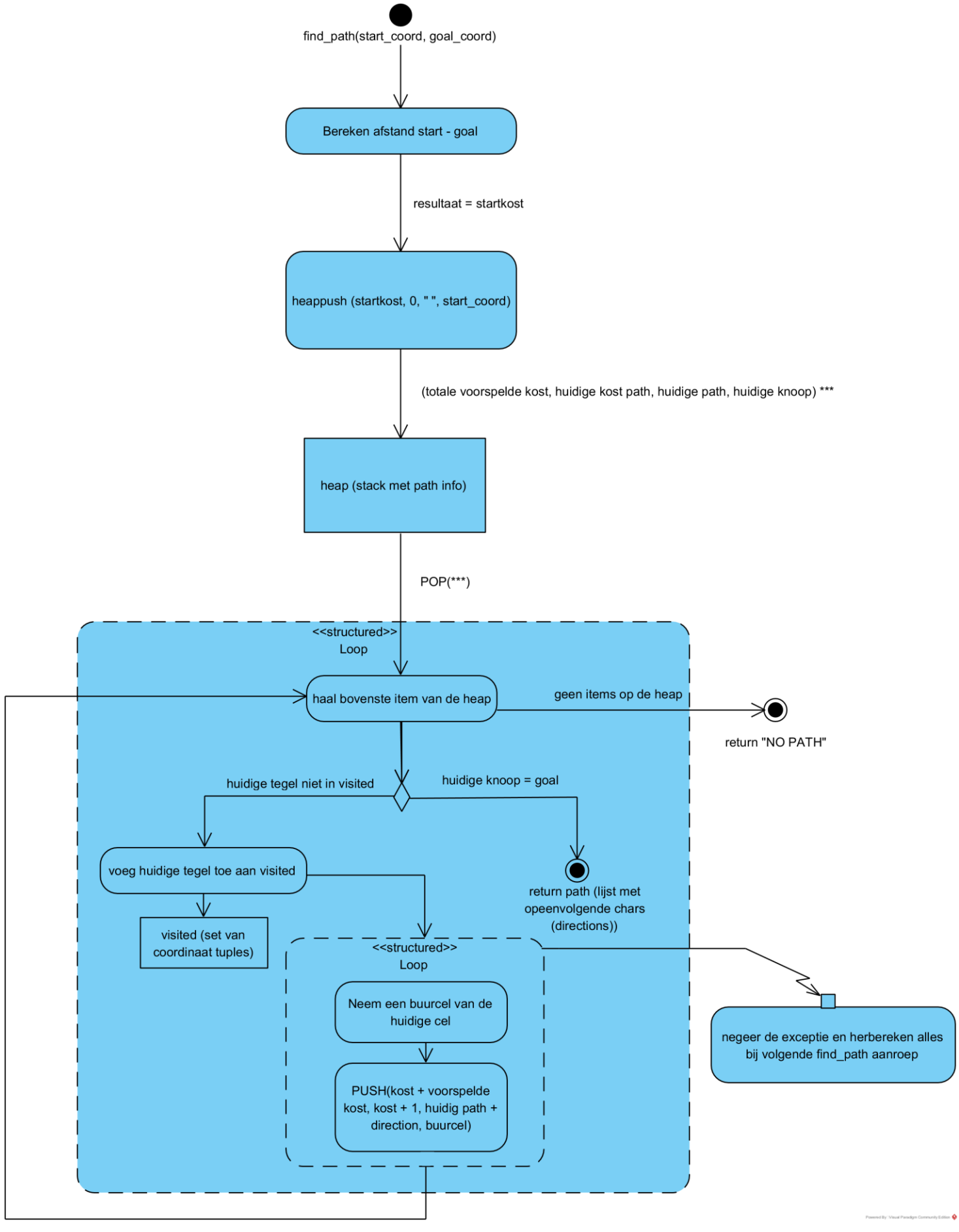
(Digitale versie zie Github)



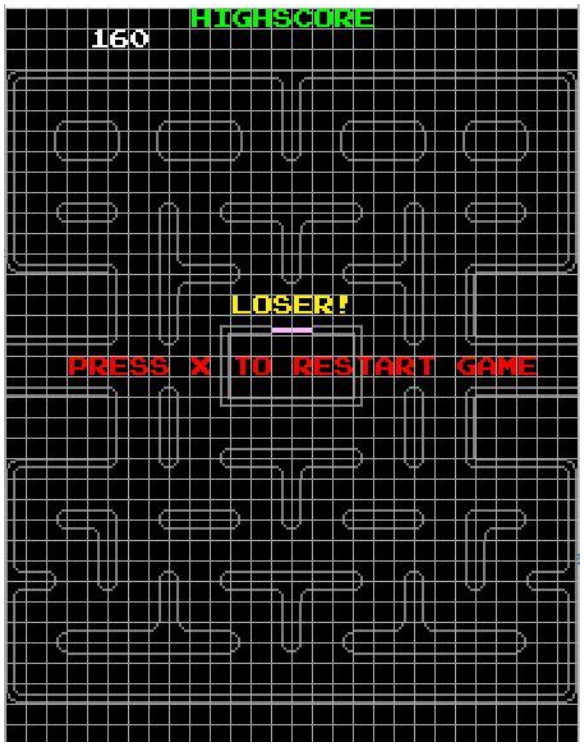
22. Bijlage 4

22.1 Activiteit diagram find_path()

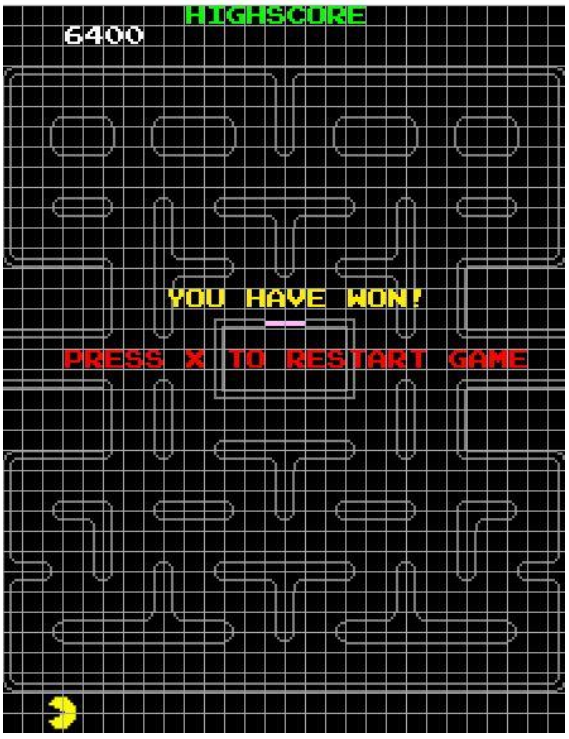
(Digitale versie zie Github)



23. Bijlage 5: Afbeeldingen



Afbeelding A



Afbeelding B