SNAKE GAME DOCUMENTATIE

Software Language 2a

Applied Design patterns (23/24)

Dorian Baies, Jari Knoop, Kim Nguyen

14 juni 2024

INHOUDSOPGAVE

1 USER STORIES

1. **Als speler wil ik graag de slang kunnen besturen, zodat ik het spel kan winnen.**
   - Als speler kan ik de slang besturen met de pijltjestoetsen of met pijltjes.
   - De slang is bestuurbaar tot dat het spel voorbij is.
   - De Slang kan niet de tegenovergestelde richting op, dat hij beweegt.

2. **Als speler wil ik logische en soepele slangbeweging:**
   - De slang beweegt zich één positie tegelijk in de richting die de speler heeft gekozen.
   - De slang beweegt met een constante snelheid die toeneemt naarmate de slang langer wordt.
   - De slang beweegt soepel over het speelveld.

3. **Als speler wil ik mijn score verhogen door appels te eten en mijn slang langer te maken:**
   - Er verschijnt willekeurig appels op het speelveld.
   - De slang kan appels eten door er overheen te bewegen.
   - Wanneer de slang voedsel eet, wordt hij langer en de score van de speler verhoogd.

4. **Als speler wil ik duidelijke spelregels zodat ik weet hoe het spel werkt:**
   - Het spel begint met een slang van een standaardlengte.
   - Het spel eindigt wanneer de slang:
     - Zichzelf raakt.
     - De rand van het speelveld raakt.
   - De score van de speler wordt weergegeven op het scherm.
   - De speler kan het spel opnieuw starten na het beëindigen.

5.1 Als speler wil ik een beginscherm om te kunnen beginnen.

- Beginscherm voor het beginnen van het spel of tijdens het pauzeren.

5.2 Als speler ik wil een scoreboard om mijn best behaalde score bij te houden.

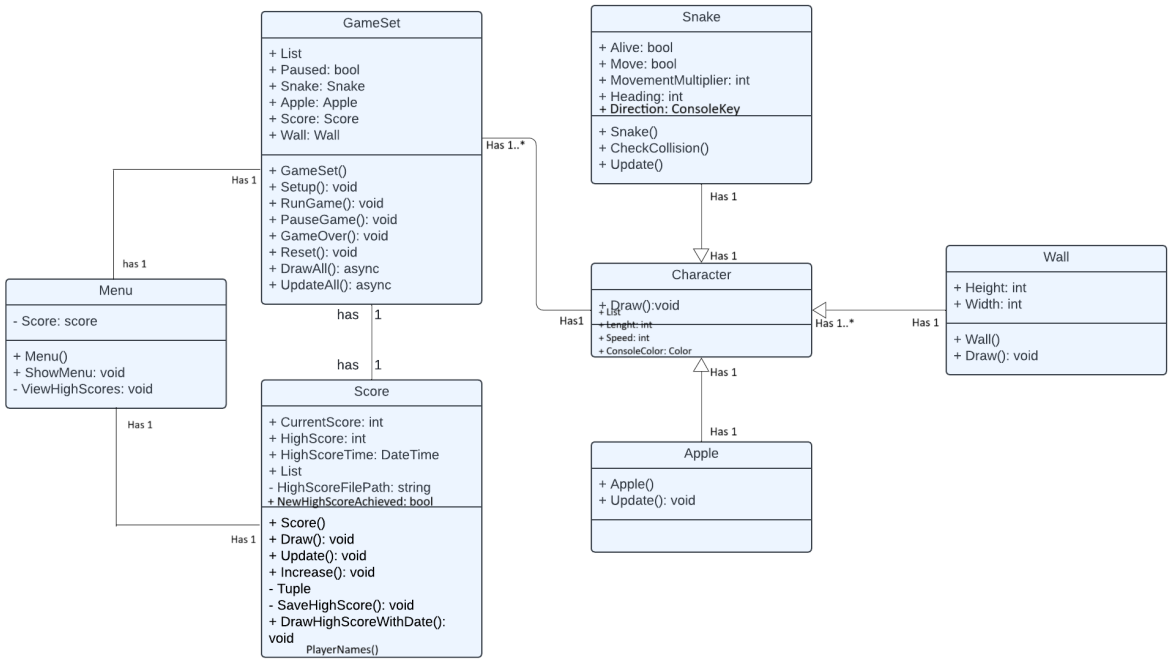
- Beginscherm om scoren op bij te houden en te verbeteren.

2 DESIGN PATTERNS

2.1 CLASS DIAGRAM

SOLID

Elke classe heft een taak. De codes zijn aanpasbaar voor uitbreiding, maar wijzigen kan niet (Oloruntoba, 2021). De class Character heeft afgeleide classen zoals Apple en Snake. De game laat alleen de interface zien die spelers nodig hebben. Classes zijn gescheiden voor abstractie, zoals dat Movement gescheiden is van Program. Ook zijn er geen foutmeldingen in het spel.



KLASSEN

- **GameSet:** Deze klasse beheert de algemene spellogica, zoals het initialiseren van het spel, het uitvoeren van de gameloop en het beëindigen van het spel.

- **Snake:** Deze klasse vertegenwoordigt de slang in het spel. Het bevat informatie over de positie van de slang, de richting waarin de slang beweegt en of de slang nog leeft.
- **Apple:** Deze klasse vertegenwoordigt de appel in het spel. Het bevat informatie over de positie van de appel.
- **Wall:** Deze klasse vertegenwoordigt de muren van het spel. Het bevat informatie over de positie van de muren.
- **Score:** Deze klasse houdt de score van de speler bij.
- **Menu:** Deze klasse toont het menu van het spel.

DE RELATIES TUSSEN DE KLASSEN

- **GameSet** heeft een **Snake**, een **Apple**, een **Wall** en een **Score**.
- **Snake** kan botsen met een **Apple** of een **Wall**.
- **GameSet** kan een nieuwe **Apple** genereren nadat de slang de huidige **Apple** heeft gegeten.
- **GameSet** kan de **Score** van de speler bijwerken wanneer de slang een **Apple** eet.
- **Menu** kan worden weergegeven wanneer het spel is gepauzeerd of beëindigd.

DE FUNCTIONALITEIT VAN DE KLASSEN

- **GameSet:**
  - **Setup()**: Initialiseert het spel door een **Snake**, een **Apple** en een **Wall** te maken.
  - **RunGame()**: Voert de gameloop uit. De gameloop controleert voortdurend voor invoer van de speler, verplaatst de slang, controleert op botsingen en tekent de gamewereld.
  - **PauseGame()**: Pauzeert het spel.
  - **GameOver()**: Beëindigt het spel en toont het game-over scherm.
  - **Reset()**: Reset het spel naar de beginwaarden.
- **Snake:**
  - **Move()**: Verplaatst de slang in de opgegeven richting.
  - **CheckCollision()**: Controleert op botsingen met een **Apple** of een **Wall**.
- **Apple:**
  - **Update()**: Verandert de positie van de **Apple** als deze door de slang is gegeten.
- **Wall:**
  - **Draw()**: Tekent de **Wall** op het scherm.
- **Score:**
  - **Increase()**: Verhoogt de score met 1.
  - **Draw()**: Tekent de score op het scherm.
- **Menu:**
  - **ShowMenu()**: Toont het menu van het spel.
  - **ViewHighScores()**: Toont de high scores van het spel.

AANVULLENDE DETAILS OVER DE KLASSEN

- De **Snake**-klasse gebruikt een lijst om de positie van de slang bij te houden. Dit maakt het mogelijk om de slang te laten groeien en krimpen.
- De **Apple**-klasse gebruikt een willekeurige generator om de positie van de **Apple** te bepalen. Dit zorgt ervoor dat de **Apple** op een willekeurige locatie op het scherm verschijnt.
- De **Wall**-klasse gebruikt een lijst om de positie van de muren bij te houden. Dit maakt het mogelijk om muren van verschillende vormen en maten te maken.
- De **Score**-klasse gebruikt een variabele om de score van de speler bij te houden. Deze variabele wordt incrementeerd wanneer de slang een **Apple** eet.
- De **Menu**-klasse gebruikt knoppen om de speler te laten kiezen tussen het spelen van het spel of het bekijken van de high scores.

SOLID

SRP:

- De klasse Snake is verantwoordelijk voor het beheren van de positie en beweging van de slang.
- De klasse Apple is verantwoordelijk voor het beheren van de positie van de appel.
- De klasse Wall is verantwoordelijk voor het beheren van de grenzen van de speelwereld.
- De klasse Score is verantwoordelijk voor het beheren van de score van de speler.
- De klasse Menu is verantwoordelijk voor het weergeven van het menu van de game.
- De klasse HighScore is verantwoordelijk voor het beheren van de highscores van de game.

OCP:

- De klasse Snake kan worden uitgebreid met nieuwe functies, zoals het toevoegen van een power-up die de slang langer maakt.
- De klasse Apple kan worden uitgebreid met nieuwe functies, zoals het toevoegen van een speciale appel die de speler extra punten geeft.
- De klasse Wall kan worden uitgebreid met nieuwe functies, zoals het toevoegen van teleporters.
- De klasse Score kan worden uitgebreid met nieuwe functies, zoals het weergeven van het aantal levens dat de speler nog heeft.
- De klasse Menu kan worden uitgebreid met nieuwe functies, zoals het toevoegen van een optie om de moeilijkheidsgraad te selecteren.
- De klasse HighScore kan worden uitgebreid met nieuwe functies, zoals het weergeven van de namen van de spelers met de hoogste scores.

LSP:

- De klasse Snake kan worden vervangen door een subklasse die een andere manier van bewegen implementeert, zoals een slang die diagonaal kan bewegen.
- De klasse Apple kan worden vervangen door een subklasse die een ander uiterlijk heeft, zoals een appel met een andere kleur of vorm.
- De klasse Wall kan worden vervangen door een subklasse die een andere vorm heeft, zoals een muur met gaten erin.
- De klasse Score kan worden vervangen door een subklasse die een andere manier van punten berekenen implementeert, zoals een score die gebaseerd is op de tijd.
- De klasse Menu kan worden vervangen door een subklasse die een andere lay-out heeft, of die andere opties weergeeft.
- De klasse HighScore kan worden vervangen door een subklasse die een andere manier van highscores opslaan implementeert, zoals het opslaan van highscores in een database.

ISP:

- De interface IDrawable kan worden opgesplitst in twee interfaces: IDrawObject en IDrawScore.
- De interface IUpdatable kan worden opgesplitst in twee interfaces: IUpdateObject en IUpdateScore.

DIP:

De klasse Game is niet afhankelijk van de concrete implementaties van de klassen Snake, Apple, Wall, Score, Menu en HighScore. In plaats daarvan is het afhankelijk van interfaces, zoals IDrawable en IUpdatable.

2.2 DESIGN SMELLS

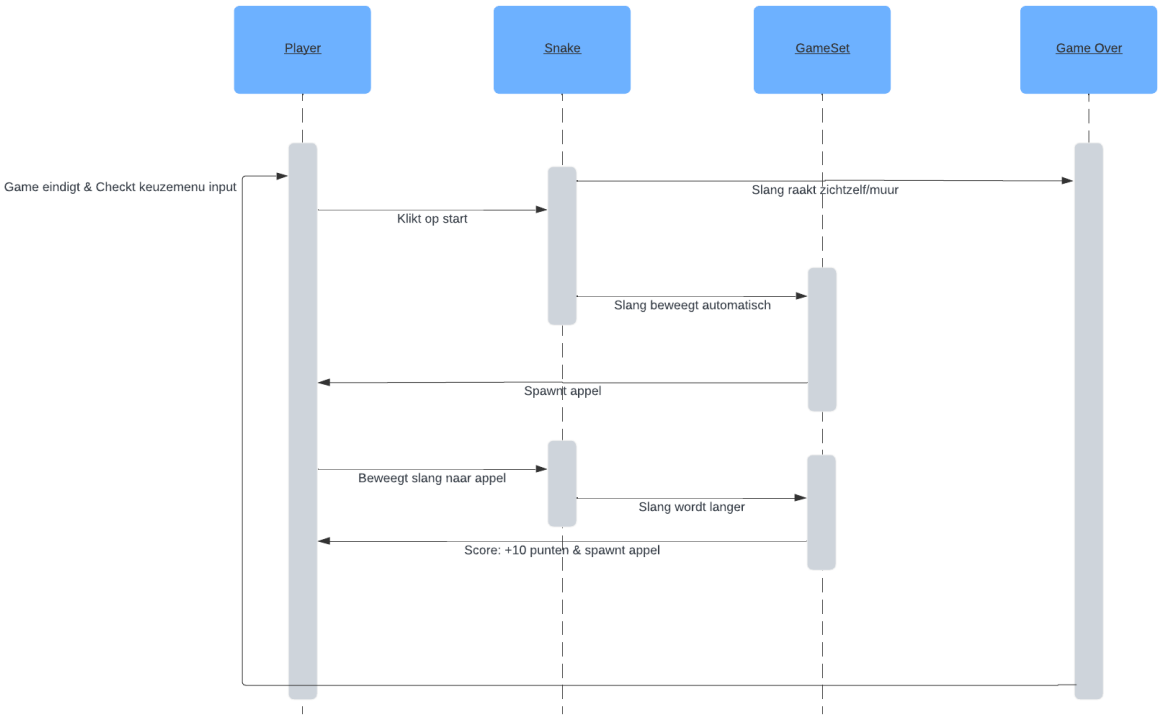
Wanneer er wordt afgeweken van fundamentele ontwerpprincipes, dan gaat de kwaliteit van het project omlaag. De leesbaarheid, testcases en uitbreidbaarheid zijn dan niet meer accuraat (Wikipedia contributors, 2024). Voor de code zijn abstracte klassen gebruikt, en ze zijn niet abstracter dan nodig.

Een aantal voorbeelden van design smells zijn:

- **Ontbrekende abstractie:** geen abstractie waar dat nodig is.
- Veelzijdige abstractie: als een Class bijvoorbeeld meerdere taken uitvoert.
- Dubbele abstractie: twee functies hebben dezelfde namen.
- Gebrekkige inkapsulatie: de toegang tot een functie is complex.
- Ongebruikte inkapsulatie: wanneer een if-else statement niet de juiste methode is om code te checken.

2.3 SEQUENCE DIAGRAM

Een diagram die laat zien hoe objecten in een systeem met elkaar communiceren. En toont de interacties tussen objecten in de volgorde waarin ze gebeuren. Je kunt precies zien welke objecten betrokken zijn, de volgorde en wat er wordt teruggestuurd.



Het diagram toont de stappen die een speler doorloopt in het spel Snake.

De stappen zijn als volgt:

- De speler drukt op **Start** om het spel te beginnen.
- De slang **beweegt automatisch** in een rechte lijn.
- De speler **drukt op de pijltjestoetsen** om de slang in een andere richting te laten gaan.
- Er **spawnt een appel** in het speelveld.
- De slang **beweegt naar de appel**.
- De **score wordt met 10 verhoogd** wanneer de slang de appel eet.
- De **slang wordt langer en sneller**.
- De slang **raakt de muur** en het spel is afgelopen.
- De slang **raakt zichzelf** en het spel is afgelopen.

Diagram toont ook de interactie tussen de speler, de slang, de gamemanager en het Game over.

De speler:

- Drukt op **Start** om het spel te beginnen.
- Drukt op de **pijltjestoetsen** om de slang te besturen.

De slang:

- Beweegt **automatisch** in een rechte lijn.
- Verandert van richting wanneer de speler op de **pijltjestoetsen** drukt.
- Wordt **langer en sneller** wanneer de slang een appel eet.
- Gaat dood wanneer de slang de muur of zichzelf raakt.

De gamemanager:

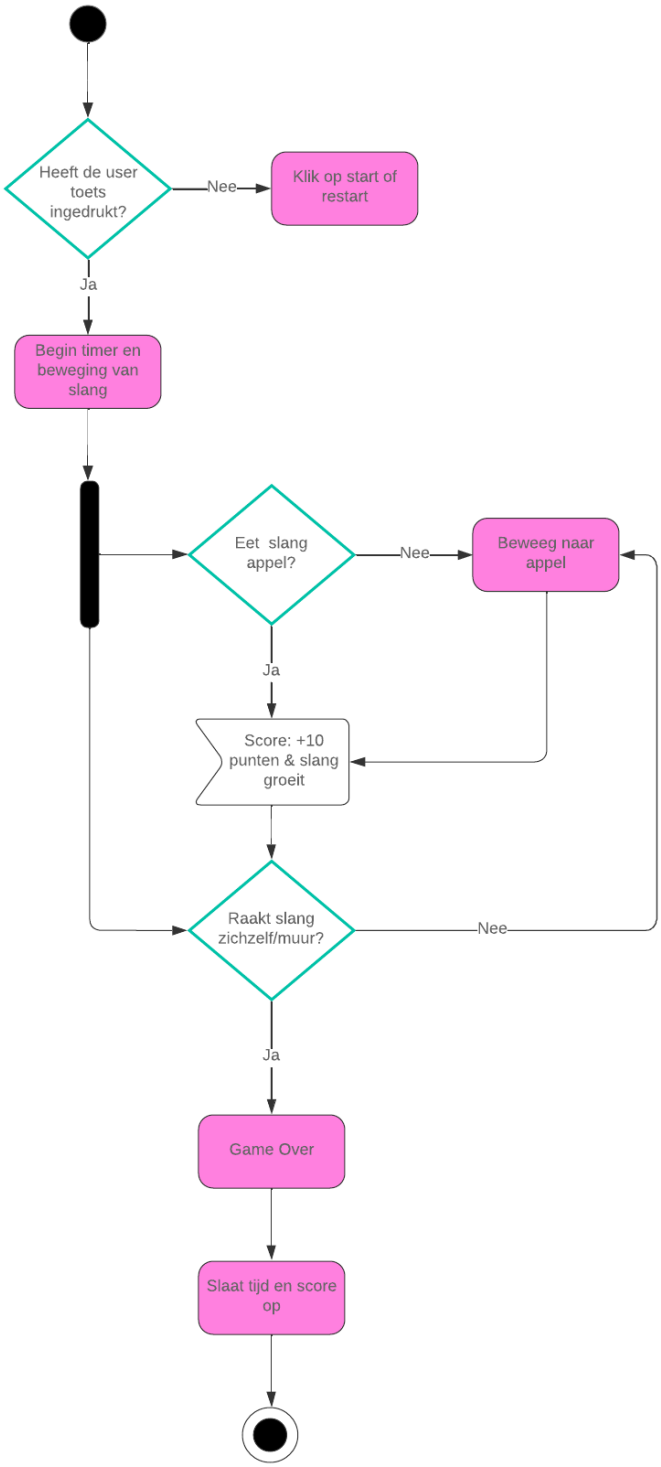
- Spawnt **appels** in het speelveld.
- Houdt de **score** bij.
- Bepaalt wanneer het spel **over** is.

Game over:

- Toont de **score** van de speler.
- Geeft de speler de mogelijkheid om het spel opnieuw te spelen.

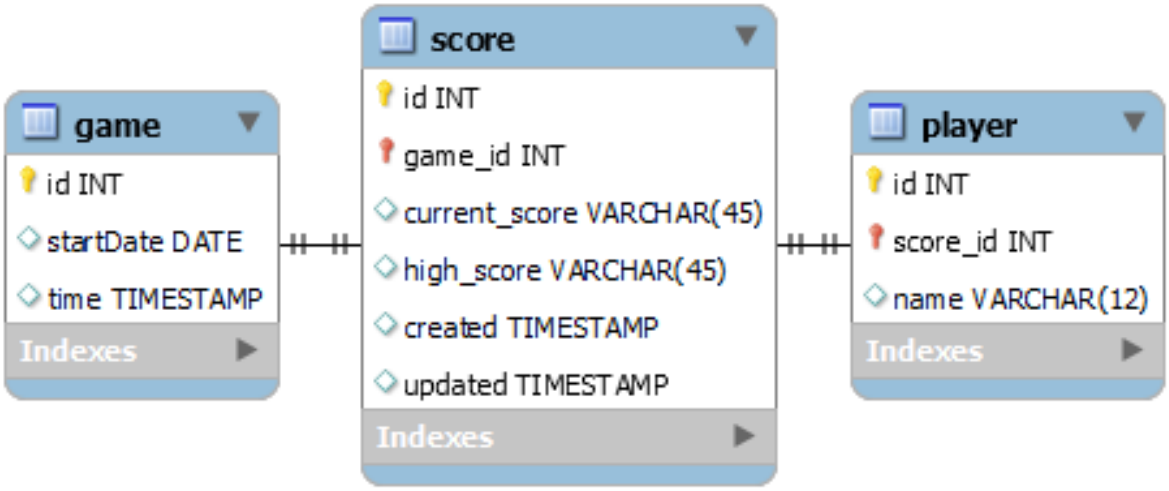
2.4 ACTIVITY DIAGRAM

De activteiten diagram is een stroomdiagram van activiteiten die in de game plaats vinden. Het toont de opeenvolgende stappen die nodig zijn voor de uitvoering van het spel. Elke stappen zijn taken die nodig zijn om te voltooien, belangrijke beslissingen en voorwaardes die binnen de game nodig zijn.



- De slang beweegt.
- Controleert of de slang een appel heeft gegeten.
    - Als de slang een appel heeft gegeten, worden er 10 punten aan de score toegevoegd en wordt de snelheid van de slang verhoogd.
- Controleert of de slang zichzelf heeft geraakt.
    - Als de slang zichzelf heeft geraakt, is het spel afgelopen.
- Controleert of de slang de muur heeft geraakt.
    - Als de slang de muur heeft geraakt, is het spel afgelopen.
- De tijd en score worden opgeslagen.
- Het spel eindigt.

2.5 ERD



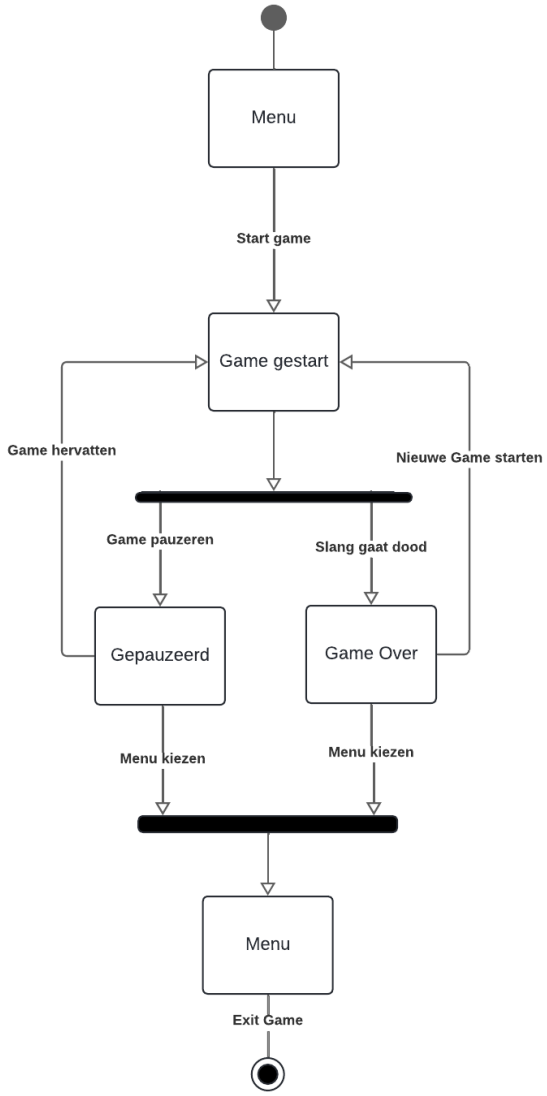
De database bevat drie tabellen:

- **Game:** Deze tabel bevat informatie over het spel, zoals de game-ID, de startdatum en de tijd.
- **Player:** Deze tabel bevat informatie over de speler, zoals de speler-ID, de naam, de hoogste score en de huidige score.
- **Score:** Deze tabel bevat informatie over de score van de speler, zoals de score-ID, de game-ID, de speler-ID en de score.

De tabellen zijn gerelateerd aan elkaar door middel van de volgende sleutelrelaties:

- **Game-Player:** De game-ID van de **Game**-tabel is gekoppeld aan de speler-ID van de **Player**-tabel. Dit betekent dat elke speler slechts één game kan spelen.
- **Game-Score:** De game-ID van de **Game**-tabel is gekoppeld aan de score-ID van de **Score**-tabel. Dit betekent dat elke game slechts één score kan hebben.
- **Player-Score:** De speler-ID van de **Player**-tabel is gekoppeld aan de score-ID van de **Score**-tabel. Dit betekent dat elke speler meerdere scores kan hebben.

2.6 STATE MACHINE DIAGRAM



De game heeft 4 statussen, die zijn gevisualiseerd in de bovenstaande State Machine Diagram.
• Menu: het menu met opties waar de gebruiker uit kan kiezen.

1. Start game
2. View high scores
3. Exit

• Game gestart: de game is begonnen en de slang beweegt

• Gepauzeerd: de game is gepauzeerd • Game Over: de game is afgelopen

• In het menu kan de gebruiker kiezen om het spel te starten of de high scores in te zien.

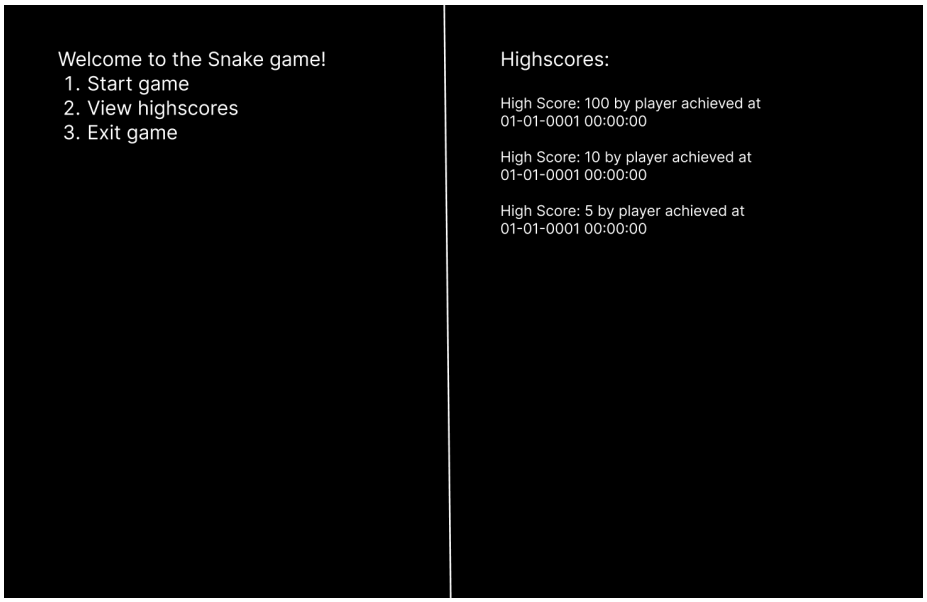
• Wanneer de speler heeft gekozen om het spel te starten, dan beweegt de slang.

• Het is Game Over wanneer de slang zichtzelf of de muur raakt.

• Dan verschijnt er “Game Over” op het scherm en kan de gebruiker ervoor kiezen om een

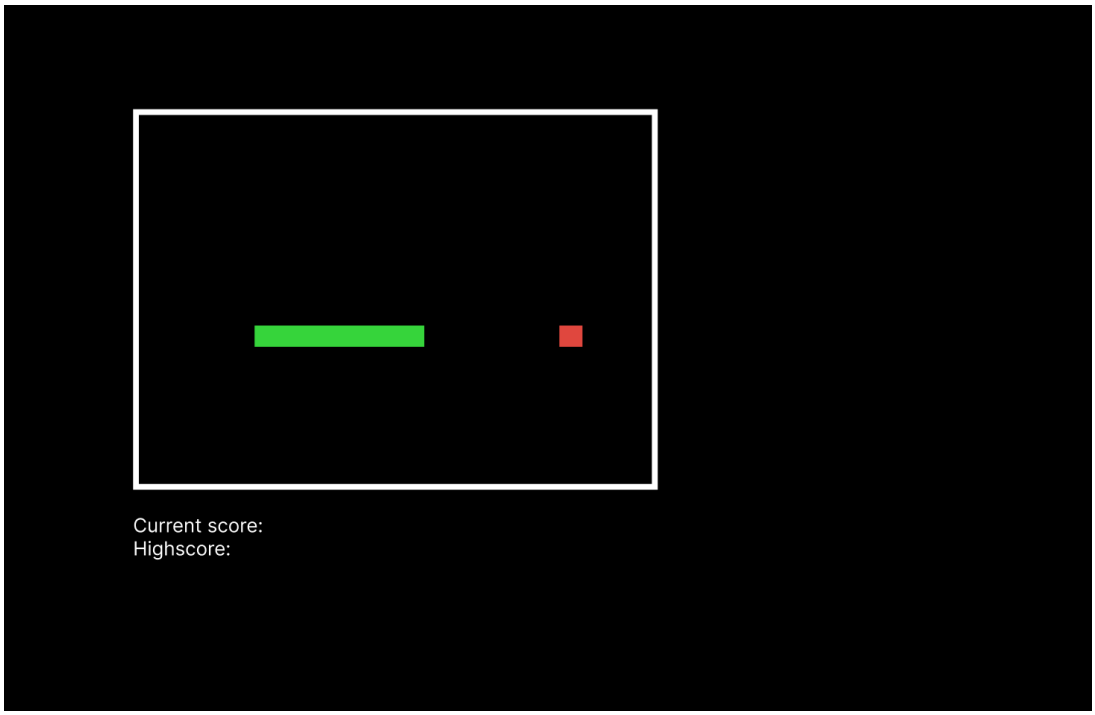
3 WIREFRAMES

HOOFDMENU



Wanneer de game voor het eerst opgestart wordt de speler verwelkomt met een menu om uit te kiezen. De opties die de gebruiker heeft is het volgende: het starten van het spel, highscores bekijken en het stoppen van het spel.

INGAME



Tet doel van het spel is om als slang de rode appels te pakken om langer en sneller te worden. De speler is af wanneer de slang in zichzelf gaat of tegen de muur aanbotst. Onder in het spel wordt de score van de speler bijgehouden en ook de highscores die behaald zijn. Wanneer een speler een nieuwe highscore zet, wordt de highscore samen met de huidige score geüpdatet.

GAME OVER

Wanneer de speler af is heeft de speler twee keuzes. De speler kan klikken op Y en opnieuw spelen met score gereset. Of kan de speler kiezen om te stoppen met spelen en wordt bedankt voor het spelen.

PAUSE

Wanneer de speler op spatie klikt kan de speler het spel pauzeren en doorgaan wanneer de speler uitkomt.

NIEUWE HIGHSCORE

Wanneer de speler een nieuwe highscore behaalt, wordt de speler gevraagd voor het opgeven van zijn of haar spelersnaam. Deze naam wordt geregistreerd met de highscore die de speler heeft behaald en is terug te vinden bij de leaderboard.

4 REFLECTIE

4.1 SPRINT

Voor het project hadden we een backlog opgesteld en dit verdeeld in 4 sprints. Een sprint duurde 1 week. Elke week deden we een paar taken samen, en aan het einde hadden we een sprintreview. Hierin reflecteerden we op wat goed en minder goed ging, ook vond de sprintplanning plaats. Aan het einde van het project vond een retrospective plaats.

SPRINT 1

Doel: Ontwerp

Refelctie: We vonden het lastig om een begin te maken aan de code, daarom waren we begonnen met de wireframes.

SPRINT 2

Doel: Classen verdeling maken en functionaliteiten opstellen

Reflectie: Het was lastig om de classen te verdelen en bepaalde functies werkten niet.

SPRINT 3

Doel: Classes en methodes aanmaken

Reflectie: De Classen goed verdelen en functies optimaliseren was lastig, omdat functies met elkaar zijn verbonden.

SPRINT 4

Doel: Features aanmaken

Reflectie: Het aanmaken van features zoals het pauzeren van de game was een grote uitdaging samen met de score functie. We hadden onvoldoende tijd voor het maken van leaderboards en de menu.

SPRINT 5 HERKANSING

Doel: Ontwerp en realisatie doelen opnieuw opstellen.

Reflectie: We merkte dat dat ontwerpkeuzes uit de 3de periode niet functioneel waren en hebben besloten om opnieuw de klassen indeling en met de juiste methodes in te delen.

SPRINT 6 HERKANSING

Doel: Classes opnieuw en logischer indelen.

Reflectie: De classes en methodes stonden in de eerste versie door elkaar waardoor het spel veel bugs bevatten. We hebben alle klassen met methodes opnieuw geschreven met als referentie van de eerste versie, waardoor de game nu beter loopt met een delta timer en nieuwe functies.

SPRINT 7 HERKANSING

Doel: Score functie verbeteren.

Reflectie: We hebben de score functie opnieuw gemaakt met hoe scores worden bijgehouden en opgeslagen.

SPRINT 8 HERKANSING

Doel: Leaderboard en Menu.

Reflectie: We hebben een nieuw menu toegevoegd aan het spel waarbij spelers een realistischer ervaring krijgen met het spelen van het spel. Ook worden de namen van de spelers die een highscore hebben behaald opgeslagen. En zijn terug te vinden in het menu bij highscores.

4.2 TEAM

We moesten een spel programmeren in C#. We kozen ervoor om Snake te programmeren. Snake is een game waarin een slang achter appels aan gaat. Hij wordt langer na het eten van elke appel.

Onze taken waren een backlog opstellen, sprints plannen, code schrijven en testen, en documentatie van de game opstellen. Het doel was op een gebruiksvriendelijk en leuke game op te leveren volgens design principes.

Eerst hebben we besloten om de Snake game te ontwikkelen, daarna de User Stories en backlog opgesteld. De wireframes werden getekend, zodat we een beeld kregen voor de code. Daarna werden de design patterns ontworpen. De weken erna schreven we de code en we hebben hem getest voor de oplevering, waarbij we elkaar hebben geholpen. Elke week hadden we een sprintreview, sprintplanning en als laatst deden we een retrospective.

Aan het einde van het project hebben we een werkende Snake game geprogrammeerd, die je kon aansturen met toetsen.

We zijn tevreden over het resultaat, omdat het werkt en het ziet er mooi uit. Ook kijken we met een positieve blik terug op de sprint, want het zorgde voor een planning en we konden alles in de juiste volgorde uitvoeren. De feedback die we van elkaar kregen heeft ons geholpen dingen beter te doen. De volgende keer zouden we het weer op deze manier doen: eerste taken opstellen, plannen, uitvoeren en daarna erop reflecteren.

CREATIONAL PATTERN

LOADINGSCORE

```
private Tuple<int, DateTime, string> LoadHighScore()
{
    if (File.Exists(HighScoreFilePath))
    {
        try
        {
            string[] scoreData = File.ReadAllLines(HighScoreFilePath);
            int score = int.Parse(scoreData[0]);
            DateTime time = DateTime.Parse(scoreData[1]);
            string name = scoreData[2];
            PlayerNames.Add(name); // Add the player name to the PlayerNames list
            return Tuple.Create(score, time, name);
        }
        catch
        {
            // Handle any errors that might occur during reading/parsing
            return Tuple.Create(0, DateTime.MinValue, string.Empty);
        }
    }
    return Tuple.Create(0, DateTime.MinValue, string.Empty);
}
```

Binnen de Score Klassen binnen ons spel is met behulp van de LoadingScore methode een eenvoudige creational pattern aan te tonen. Deze methode is verantwoordelijk voor afhankelijk of er een highscorebestand bestaat en of het correct te lezen kan worden, een Tuple objecten wordt gecreëerd dat de highscore, de datum en spelersnaam bevat. Als er geen bestand is of als het lezen mislukt, wordt een standaardwaarde (0 voor de score, een minimale datum en een lege naam) teruggegeven.

Betreft creational pattern beschrijft deze methode de volgende concepten:

- Voorwaardelijke creatie: De objectcreatie is afhankelijk van bepaalde voorwaarden.
- Verbergen van complexiteit: De details van hoe het object wordt gemaakt (inclusief foutafhandeling) zijn verborgen binnen de methode.

APPEL

```
public class Apple : Character {
    public Apple(ConsoleColor color) { //Constructor to give an apple a CUSTOM COLOUR!!!
        Color = color;
        Speed = 0;
        Length = 1;
    }
```

Bijbehorende design pattern: Creational Pattern

Deze code bevat een constructor die elke object zijn eigen specifieke start kleur kan geven, deze is zo gemaakt met het idee dat bij mogelijke uitbreidingen van het spel er verschillende varianten appels gemaakt zouden kunnen worden. Dit valt onder de Factory Method design pattern ook al is de uitwerking daarvan nog simpel.

FACTORY METHOD

```
public GameSet(Score score) {
    this.Score = score;
    Wall = new(60, 20);
    Snake = new(Wall.Width/2, Wall.Height/2, ConsoleColor.DarkGreen);
    Apple = new(ConsoleColor.Red);
    Wall.Draw();
}
```

Factory Method Pattern

De GameSet is een constructor voor de Wall, Snake en Apple class. Deze worden aangemaakt wanneer de GameSet class wordt aangeroepen.

BEHAVIOURAL PATTERN

GAMESET UPDATE EN INCREASE

```
public void Update(GameSet gameSet)
{
    gameSet.DrawPositions.Add((Position.First().x, Position.First().y, this));
    if (CurrentScore > HighScore)
    {
        HighScore = CurrentScore;
        HighScoreTime = DateTime.Now;
        NewHighScoreAchieved = true;
    }
}

public void Increase(Snake snake)
{
    CurrentScore += snake.Speed;
    snake.Speed++;
}
```

In de gegeven Score klasse zie je gedragspatronen in de Update en Increase methoden. Deze methoden laat de interactie en verandering zien van objecten op basis van bepaalde gebeurtenissen of condities zoals: als de huidige score hoger is dan de highscore, verandert de huidige score in de nieuwe highscore.

- Subject: Het ‘Score’ object.
- Observers: Objecten zoals ‘GameSet’ en ‘Snake’ die geïnteresseerd zijn in wijzigingen van de ‘Score’.

Het Score object (Subject) geeft aan de huidige scorepositie het GameSet object (Observer) door deze toe te voegen aan positions. Dit houdt in dat het GameSet object op de hoogte wordt gesteld van de nieuwe positie en score informatie.

SLANG BEWEGING

```
public void Update(GameSet gameSet) {
    if(Alive && Move) {
        Heading = Position.Last();
        MovementMultiplier = 1000/Speed;

        if(Console.KeyAvailable) {
            ConsoleKey key = Console.ReadKey(true).Key;
            switch(key) { //Switch statement prevents movement in the opposite direction to prevent suicide
                case ConsoleKey.LeftArrow:
                    if(Direction != ConsoleKey.RightArrow) { Direction = key; };
                    break;
                case ConsoleKey.RightArrow:
                    if(Direction != ConsoleKey.LeftArrow) { Direction = key; };
                    break;
                case ConsoleKey.UpArrow:
                    if(Direction != ConsoleKey.DownArrow) { Direction = key; };
                    break;
                case ConsoleKey.DownArrow:
                    if(Direction != ConsoleKey.UpArrow) { Direction = key; };
                    break;
            }
        }
```

```
switch(Direction) {
    case ConsoleKey.RightArrow:
        Heading.x++;
        break;
    case ConsoleKey.LeftArrow:
        Heading.x--;
        break;
    case ConsoleKey.DownArrow:
        Heading.y++;
        MovementMultiplier *= 2; //Moves slower going vertically
        break;
    case ConsoleKey.UpArrow:
        Heading.y--;
        MovementMultiplier *= 2; //Moves slower going vertically
        break;
}
```

Bijbehorende design pattern: Behavioural Pattern

De code voor de beweging van de slang bevat onder andere een State Pattern die gebaseerd is op de eigenschappen ‘Alive’ en ‘Move’ van de slang, als de slang niet in deze staat verkeert kan deze niet bewegen. Voor de beweging zelf is er een Command Pattern, deze vangt de user’s inputs op en door middel van een switch statement zorgt hij dat de correcte bijbehorende actie wordt uitgevoerd.

PAUSEGAME

```
public void PauseGame() {
    if(Console.KeyAvailable) {
        ConsoleKey Key = Console.ReadKey(true).Key;
        if(Key == ConsoleKey.P || Key == ConsoleKey.Spacebar) {
            if(Paused == false) {
                Paused = true;
            }
            else {
                Paused = false;
            }
        }
    }
}


public void GameOver()
{
    string gameOverText = "YOU DIED";
    string continueText = "CONTINUE? Y/N";
    Console.SetCursorPosition((Wall.Width / 2 - (gameOverText.Length / 2)), Wall.Height / 2 - 1);
    Console.ForegroundColor = ConsoleColor.White;
    Console.Write(gameOverText);
    Console.SetCursorPosition((Wall.Width / 2 - (continueText.Length / 2)), Wall.Height / 2); // New line so
walls dont break on writing line
    Console.Write(continueText);
    if (Score.NewHighScoreAchieved)
    {
        Console.Write("\nCongratulations! You've achieved a new high score. Please enter your name: \n");
        string playerName = Console.ReadLine();
        Score.PlayerNames.Add(playerName);
```

```
            Score.SaveHighScore();
            Score.NewHighScoreAchieved = false;
        }
        if(Console.KeyAvailable) {
            ConsoleKey Key = Console.ReadKey(true).Key;
            Reset(Key);
        }
    }
```

Observer Pattern:

PauseGame, GameOver, Paused boolean en Snake.Alive boolean kijken naar de user input en werken de game bij naar de staat van de method. Zoals dat de game gepauzeerd kan worden en je kan kiezen om de game opnieuw te spelen nadat de game is afgelopen.

CONCURRENCY PATTERN

GAMESET DRAWALL

```
public async Task DrawAll() {
    foreach(var (x, y, o) in DrawPositions) {
        Console.SetCursorPosition(x, y);
        if(o == Snake && Snake.Alive) {
            Console.BackgroundColor = Snake.Color;
            Snake.Draw();
        }
        if(o == Apple) {
            Console.BackgroundColor = Apple.Color;
            Apple.Draw();
        }
        if(o == Score) {
            Snake.Move = false;
            Console.ForegroundColor = ConsoleColor.White;
            Console.BackgroundColor = ConsoleColor.Black;
            Score.Draw();
        }
    }
    DrawPositions.Clear();
    Console.BackgroundColor = default;
    Console.ForegroundColor = default;
    Snake.Move = true;
    await Task.Delay(Snake.MovementMultiplier / 2);
}
```

De DrawAll methode gebruikt het Asynchronous Programming Pattern als een concurrency pattern. Dit patroon zorgt ervoor dat de methode asynchroon en parallel worden uitgevoerd. Dit is cruciaal voor het soepel laten verlopen van het spel. De combinatie van async, await, en Task.Delay in de klassen van parallelle taakuitvoering in de RunGame methode weergeeft een effectieve toepassing van concurrency in ons project. In de RunGame methode wordt DrawAll parallel uitgevoerd met UpdateAll en een vertragingstaak.

Loop Through Draw Positions: De foreach loop doorloopt alle items in de DrawPositions lijst. Elk item bevat de x- en y-coördinaten en het object dat getekend moet worden.

GAMESET UPDATEALL

```
public async Task UpdateAll(int deltaTime) {
    Snake.Update(this);
    Apple.Update(this);
    Score.Update(this);
    await Task.Delay(Snake.MovementMultiplier / 2 + deltaTime);
}
```

Bijbehorende design pattern: Concurrency Pattern

De UpdateAll methode gebruikt een Task-Based Asynchronous Pattern, dankzij de await kunnen deze methodes op hun eigen tempo voltooid worden en blokkeren ze niet de main thread waar de game op draait.

GAMESET RUNGAME

```
public void RunGame() {
    while(Paused == false) {
        Task.WaitAny(
            DrawAll(),
            UpdateAll(),
            Task.Delay(Snake.MovementMultiplier)
        );
        //PauseGame();
        if(!Snake.Alive) {
            GameOver();
        }
    }
}
```

Task Parallelism:

In Task.WaitAny zijn er 3 methodes: DrawAll(), Update(), Task.Delay(Snake.MovementMultiplier). Deze methodes kunnen parallel worden uitgevoerd dankzij Task.WaitAny.

STRUCTURAL PATTERN

GAMSET

```
public class GameSet {
    public List<(int x, int y, object o)> DrawPositions = [];
    public bool Paused = false;
    public Snake Snake;
    public Apple Apple;
    public Score Score;
    public Wall Wall;

    public GameSet(Score score) {
        this.Score = score;
        Wall = new(60, 20);
        Snake = new(Wall.Width / 2, Wall.Height / 2, ConsoleColor.DarkGreen);
        Apple = new(ConsoleColor.Red);
        Wall.Draw();
    }
}
```

De GameSet klasse beheert een verzameling game-componenten (Snake, Apple, Wall, Score). Deze componenten kunnen worden gebruikt als onderdeel van een de algmene structuur waarbij GameSet de samenstelling is die zijn onderliggende componenten bevat en beheert.

FACADE PATTERN

```
public class GameSet {
    public List<(int x, int y, object o)> DrawPositions = [];
    public bool Paused = false;
    public Snake Snake;
    public Apple Apple;
    public Score Score;
    public Wall Wall;

    public GameSet() ...

    public void Setup() ...

    public void RunGame() ...

    public void PauseGame() ...

    public void GameOver() ...

    public void Reset(ConsoleKey Key) ...

    public async Task DrawAll(int deltaTime) ...

    public async Task UpdateAll(int deltaTime) ...
}
```

Bijbehorende design pattern: Structural Pattern

De GameSet klasse bevat een Facade pattern, gebruikers krijgen dankzij deze klasse een makkelijk te benaderen toegangspunt tot de applicatie. Dankzij het samenvoegen van alle methodes tot één gangbare klasse is de overzichtelijkheid en mogelijkheid voor onderhoud ook versimpeld.

VIEWHIGHSCORE

```
namespace RealSnakeGame;


public class Menu : Score
{
    private Score score;


    public Menu(Score score)
    {
        this.score = score;
    }
    public void ShowMenu()
    {
        while (true)
        {
            Console.WriteLine("Welcome to the Snake game!");
            Console.WriteLine("1. Start game");
            Console.WriteLine("2. View high scores");
            Console.WriteLine("3. Exit");
            Console.Write("Enter your choice: ");


            string choice = Console.ReadLine();


            switch (choice)
            {
                case "1":
                    // Clear the console
                    Console.Clear();


                    // Start the game
```

```
                GameSet gameSet = new GameSet(score);
                gameSet.RunGame();
                break;
            case "2":
                // View high scores
                ViewHighScores();
                break;
            case "3":
                // Exit the application
                Environment.Exit(0);
                break;
            default:
                Console.WriteLine("Invalid choice. Please try again.");
                break;
        }
    }
}

private void ViewHighScores()
{
    Console.Clear();
    score.DrawHighScoreWithDate();
    Console.WriteLine("\nPress any key to return to the menu...");
    Console.ReadKey();
    Console.Clear();
}
}
```

Facade Pattern

De ShowMenu methode van de Menu class maakt een menu aan voor de user, zodat de user een keuze kan maken. `ViewHighScores` method weergeeft de highscores van de user.

Composition

De Menu class heeft een GameSet class, en deze wordt aangeropen als de user kiest voor optie “1”. Ook gebruikt de Menu class GameSet en Score class. Hierdoor kan de menu en scores worden weergegeven en de game worden gestart.

REFERENTIES

Oloruntoba, S. (2021, 30 november). *SOLID: The First 5 Principles of Object Oriented Design*. DigitalOcean. https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

Wikipedia contributors. (2024, 29 maart). *Design smell*. Wikipedia. https://en.wikipedia.org/wiki/Design_smell#:~:text=In%20computer%20programming%2C%20a%20design,negatively%20impact%20the%20project's%20quality

UML Class Diagram Tutorial. (z.d.). Lucidchart. https://www.lucidchart.com/pages/uml-class-diagram

Wikipedia contributors. (2024, 10 maart). *Entity–relationship model*. Wikipedia. https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model