SNAKE GAME DOCUMENTATIE

Software Language 2a & Applied Design Patterns

Dorian Baies, Jari Knoop, Kim Nguyen

5 april 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 de letters W,A S en D.
   - 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 UML

UML staat voor Unified Modeling Language. De taal die gebruikt wordt om complexe systemen in kaart te brengen en overzichtelijk uit te leggen. UML helpt met het structuur en gedrag van software begrijpen en visualiseren.

Er zijn verschillende soorten UML-diagrammen, waaronder:

- **Klassendiagrammen:** Het tonen van de relaties van de klassen in een software.
- **Use Case Diagram:** Laten zien hoe de gebruikers met een systeem kunnen interacteren. Een visuele weergave van de functionaliteiten het systeem en hoe de gebruiker daarmee om kan gaan.
- **Activity Diagram:** Een stroomdiagram die de stappen en taken in het proces worden uitgevoerd weergeeft. Dit wordt weergeven als afgeronde rechthoeken.
- **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.
- **State Machine Diagram:** State machine diagram wordt ook wel het toestandsdiagram genoemd. En laat de verschillende toestanden en overgangen zien waar objecten zich kunnen vinden.

In het kort. UML is een hulpmiddel voor het beter ontwerpen en ontwikkelen van software. Het kan helpen de complexiteit van systemen gemakkelijk uit te leggen en in kaart te brengen. Ook kunnen de diagrammen de relaties tussen objecten en klassen aantonen. Dit helpt voor een betere ontwikkeling en comminucatie binnen het team, en klanten en ontwikkelaar.

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.

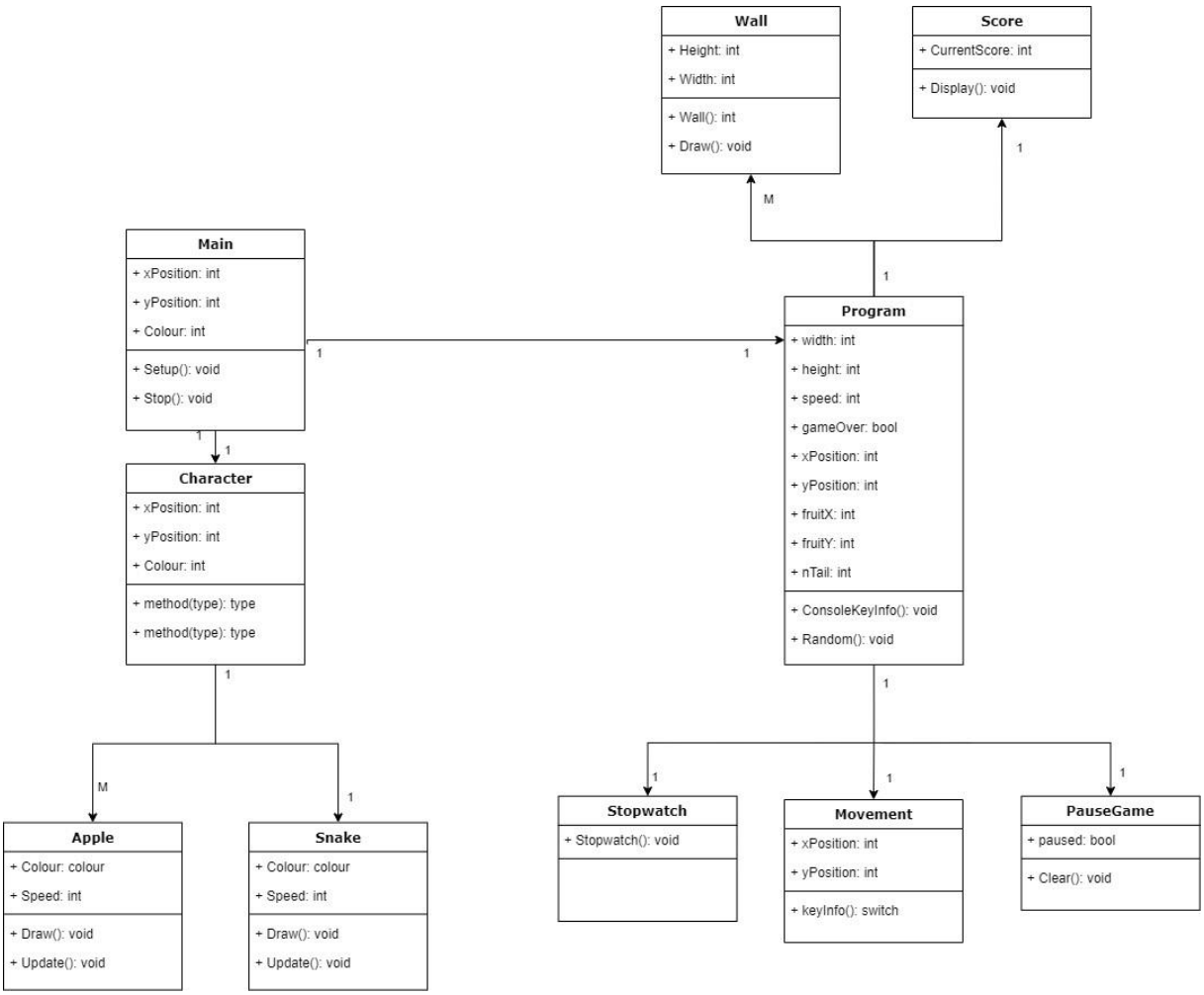
Een aantal voorbeelden van design smells zijn:

- **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 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.



De Class Diagram weergeeft de structuur van de Game. Het laat verschillende Classen zien en hun relaties.

Deze Class Diagram toont de volgende Classen:

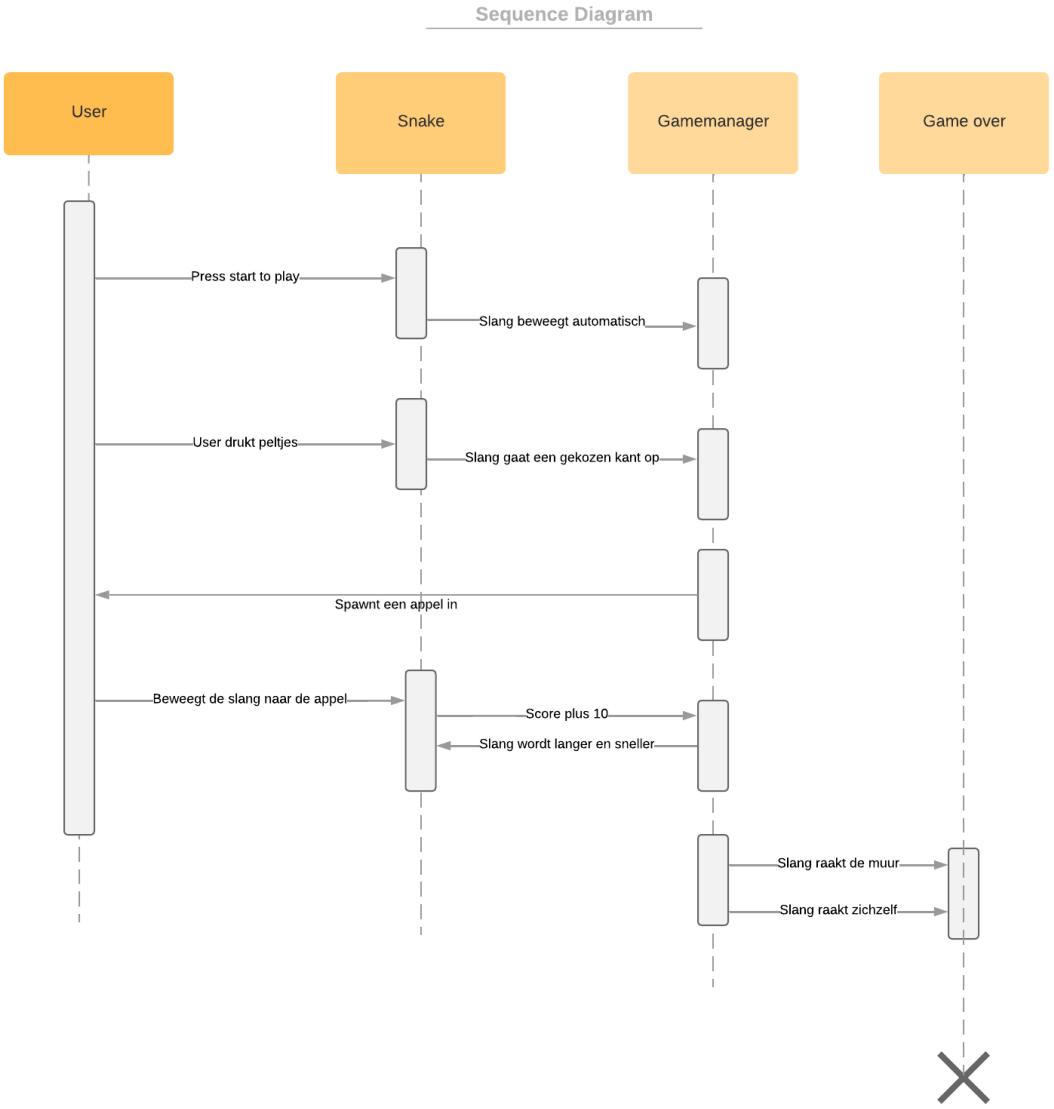
- Wall: het speelveld heeft randen
- Snake: het spelobject dat die de speler kan besturen
- Apple: het voedsel van de slang
- Stopwatch: dit is om de tijd van de speler bij te houden
- Movement: dit bepaalt de beweging van de slang
- PauseGame: deze functie is om het spel te pauzeren

Hieronder worden de relaties tussen de Classen beschreven:

- De x- en y-coördinaten bepalen de positie van de slang.
- De snelheid bepaald hoe snel de slang beweegt.
- De slang groeit door appels te eten.
- Het speelveld hangt af van hoe hoog en breed de muren zijn.
- Apples verschijnen op willekeurige plaatsen in het speelveld.
- De tijd van de stopwatch wordt in seconden bijgehouden.
- De slang beweegt in een bepaalde richting.
- De status geeft aan of het spel doorgaat of is gepauzeerd.

2.4 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:

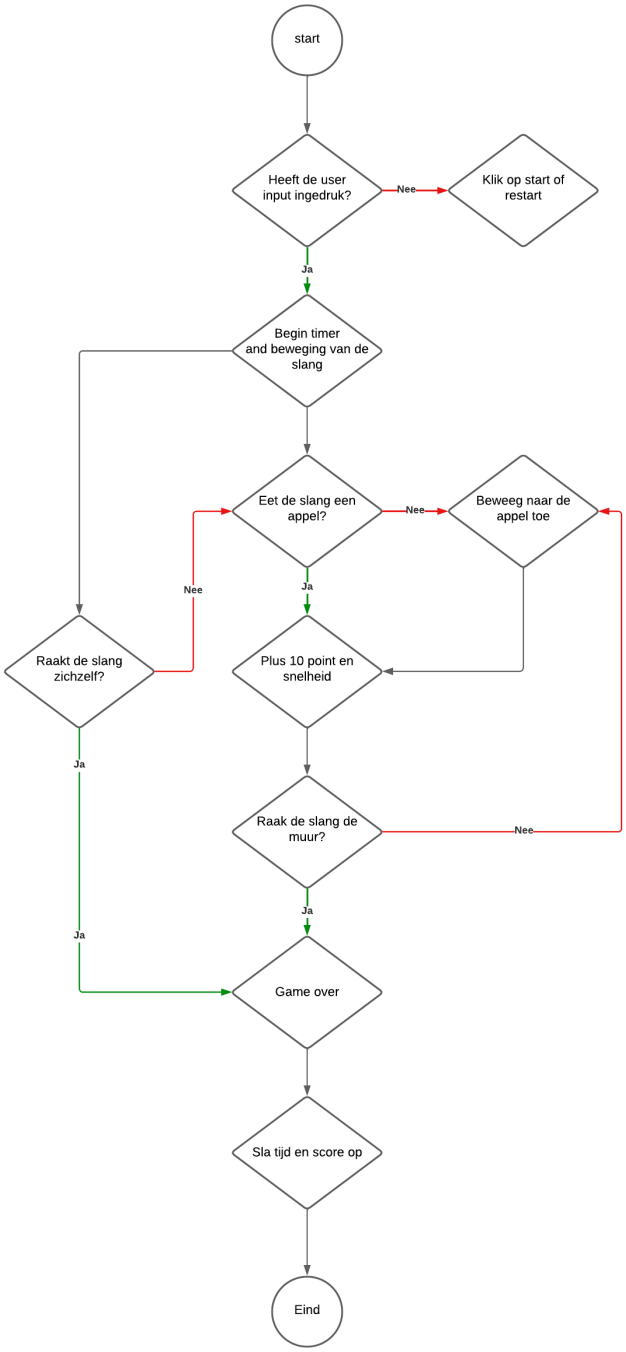
- Spawnt **appels** in het speelveld.
- Houdt de **score** bij.
- Bepaalt wanneer het spel **over** is.

Game over:

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

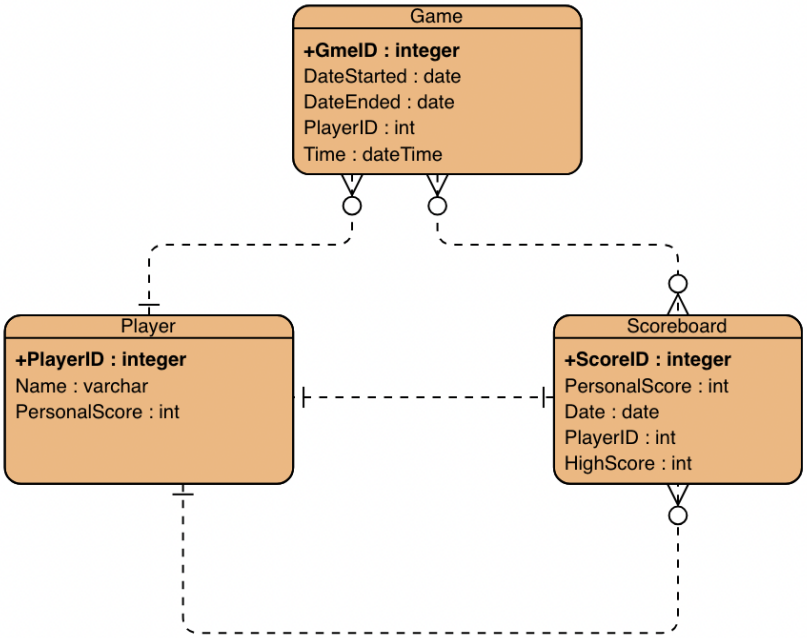
2.5 Activity Diagram

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



Een ERD, ofwel Entity-Relationship Diagram, is een visuele weergave van de structuur van een database. Het toont de verschillende entiteiten (tabellen) in de database en hoe ze met elkaar zijn gerelateerd.

- **Game:** Deze tabel bevat informatie over games, zoals de startdatum, einddatum, en de ID van de speler die het spel heeft gespeeld.
- **Player:** Deze tabel bevat informatie over spelers, zoals hun ID, naam en persoonlijke score.
- **Scoreboard:** Deze tabel bevat informatie over de scores van spelers op het scorebord, inclusief de ID van de speler, de datum en de highscore.

De relaties tussen de tabellen zijn als volgt:

- **Game:** Een spel kan worden gespeeld door een speler.
- **Player:** Een speler kan een highscore hebben op het scorebord.
- **Scoreboard:** Een score op het scorebord is van een speler.

3 Concurrency Pattern

Concurrency Patterns wordt gebruikt in programmeren, om klasses te schrijven die meerdere taken kunnen uitvoeren.

- De muren worden getekend wanneer de game is gestart.

```
//Simpele class voor muur tekenen
public class Wall{
    public int Height, Width;

    public Wall(int width,int height) {
        Height = height + 2;
        Width = width + 2;
    }

    public void Draw() {
        for(int h = 0; h <= Height; h++) {
            for(int w = 0; w <= Width; w++) {
                if((w == 0 || w == Width) && !(h == 0 || h == Height)) {
                    Console.Write("\u2551");
                }
                if(!(w == 0 || w == Width) && (h == 0 || h == Height)) {
                    Console.Write("\u2550");
                }
                if(!(w == 0 || w == Width) && !(h == 0 || h == Height)) {
                    Console.Write(" ");
                }
                if(w == 0 && h == 0) {
                    Console.Write("\u2554");
                }
                if(w == Width && h == 0) {
                    Console.Write("\u2557");
                }
                if(w == 0 && h == Height) {
                    Console.Write("\u255A");
                }
                if(w == Width && h == Height) {
                    Console.Write("\u255D");
                }
            }
            Console.Write("\n");
        }
    }
}
```

- De slang groeit wanneer hij een appel eet.

```
public class Snake : Character {
    //Constructor voor slang, Wall om te kiezen tussen welke wall hij komt en
kleur omdat we allemaal andere kleuren slang hebben
    public Snake(Wall wall, ConsoleColor colour) {
        Colour = colour;
        Speed = 8;
        Length = 1;
        StartPosition = (wall.Width / 2, wall.Height / 2);
    }

    //Tekent de slang in slangenkleur en tekent over zijn spoor
    public void Draw(Snake snake) {
        if(snake.xPosition == null && snake.yPosition == null) {
            Console.SetCursorPosition(StartPosition.x, StartPosition.y);
```

```
        }
        if(snake.xPosition != null && snake.yPosition != null){
            Console.SetCursorPosition(snake.xPosition[0], snake.yPosition[0]);
        }
        Console.ForegroundColor = snake.Colour;
        Console.Write("\u2558");
        Console.ResetColor();

        //Omdat je de console niet elke keer meer cleart moet je dit handmatig
        doen door naar de slang zijn staart
        //Te gaan en het handmatig over te kleuren. Vergeet niet de cursor terug
        te doen.
        if(snake.xPosition != null && snake.xPosition.Count() > snake.Length){
            (int x, int y) currentPosition = (snake.xPosition[0],
snake.yPosition[0]);
            Console.SetCursorPosition(snake.xPosition[snake.Length],
snake.yPosition[snake.Length]);
            Console.ForegroundColor = ConsoleColor.Black;
            Console.Write("\u2558");
            Console.ResetColor();
            Console.SetCursorPosition(currentPosition.x, currentPosition.y);
        }
    }
```

- Tegelijkertijd gaat de score omhoog.

```
public class Score{
    public int CurrentScore = 0;
    //Dictionary<string, int> Leaderboard = [];
          //Dictionary kan gebruikt worden voor een leaderboard.
      Leadboard.Add("Naam", Score);

    public void AddToLeaderboard(Score score){
              //Gebruiker krijgt de mogelijkheid om zijn naam in te vullen met
      zijn huidige score.
        //"Vul je naam in op op het leaderboard te komen!"
        //Leaderboard.Add(Console.ReadLine(), score.CurrentScore);
    }

    public void Display(){
        //Insert behaviours for current score and high score
    }

    public void Reset(){
        //Wanneer de game opnieuw start
        //score.currentScore = 0;
    }
}

//Als de slang positie over een appel positie gaat, meer lengte, meer snelhid,
meer punten, nog meer punten hoe sneller je bent!
if(apple.xPosition == snake.xPosition && apple.yPosition == snake.yPosition) {
    snake.Length++;
    snake.Speed++;
    //Gebaseerd op de slang zijn snelheid krijgt hij meer punten
    score.CurrentScore += (10+(10*(snake.Speed/100)));
}

public class Score{
    public int CurrentScore = 0;
    //Dictionary<string, int> Leaderboard = [];
```

```
    //Dictionary kan gebruikt worden voor een leaderboard. Leadboard.Add("Naam",
Score);

    public void AddToLeaderboard(Score score){
        //Gebruiker krijgt de mogelijkheid om zijn naam in te vullen met zijn
huidige score.
        //"Vul je naam in op op het leaderboard te komen!"
        //Leaderboard.Add(Console.ReadLine(), score.CurrentScore);
    }

    public void Display(){
        //Insert behaviours for current score and high score
    }

    public void Reset(){
        //Wanneer de game opnieuw start
        //score.currentScore = 0;
    }
}
```

- De appel wordt gegenereerd op willekeurige plaatsen wanneer de slang beweegt.
  Er komt gelijk een nieuwe appel als de vorige appel is opgegeten door de slang.

```
public class Apple : Character {
    //Lijst van appels zodat ze makkelijk bijgehouden kunnen worden
    public static List<Apple> Apples = [];

    //Een constructor voor een appel, enige parameter is kleur
    public Apple(Wall wall, ConsoleColor colour){
        Colour = colour;
        Speed = 0;
        Length = 1;
        StartPosition = (new Random().Next(1, wall.Width - 1), new
Random().Next(1, wall.Height - 1));
    }

    //Tekenfunctie voor appel, als hij aangeroepen wordt zoekt hij een
    willekeurige plek om een appel te tekenen en voeg hem toe aan de appel lijst!
    public void Draw(Wall wall, Apple apple) {
        apple.xPosition[0] = new Random().Next(1, wall.Width - 1);
        apple.yPosition[0] = new Random().Next(1, wall.Height - 1);
        Console.SetCursorPosition(apple.xPosition[0], apple.yPosition[0]);
        Console.ForegroundColor = apple.Colour;
        Console.Write("\u2558");
        Console.ResetColor();
        Apples.Add(apple);
        //Ze de cursor terug naar die waar de slang zou zijn
        //Console.SetCursorPosition(oude positie);
    }

    public void Update(Wall wall, Apple apple) {
        //Als er meer dan 1 appel zou zijn, verwijder de appel uit de appel lijst
        if(Apples.Count() > 1) {
            Apples.Remove(apple);
        }
        //Als er geen appels zijn, roep de Draw() methode aan
        if(Apples.Count() == 0) {
            Draw(wall, apple);
        }
    }
}
```

- De stopwatch loopt zolang de slang beweegt.

```
public class Game {
    public bool gameOver = false;
    Stopwatch timer = new();

}
```

- Wanneer de slang een muur of zijn staart raakt, eindigt het spel.

```
public void GameOver(Snake snake, Wall wall, Score score) {
     if (snake.xPosition[0] < 0 || snake.xPosition[0] >= wall.Width ||
snake.yPosition[0] < 0 || snake.yPosition[0] >= wall.Height){
         gameOver = true;
     }

     // Checkt of de slang tegen zichzelf aan botst
     for (int i = 1; i < snake.Length; i++){
         if (snake.xPosition[0] == snake.xPosition[i] && snake.yPosition[0] ==
snake.yPosition[i]){
             gameOver = true;
         }
     }
     timer.Stop();
     Console.WriteLine($"Score: {score.CurrentScore}");
     Console.WriteLine($"Time: {timer.Elapsed.ToString(@"mm\:ss")}");
 }
Game Game = new();
while(!Game.gameOver) {
    snake.Draw(snake);
    apple.Draw(wall, apple);
    snake.Update(snake, apple, currentScore, Console.ReadKey());
    apple.Update(wall, apple);
    Game.Pause(Console.ReadKey());
}
```

4 Creational Pattern

Creational Patterns gaan over hoe objecten worden aangemaakt. Deze patronen bevatten opties voor het instantiëren van objecten en ook mogelijkheden voor de beste manieren voor het aanmaken daarvan. Creational Patterns kunnen helpen bepalen hoe uitgebreid het aanmaken van een object zal worden, ook helpt het met onder andere het opnieuw gebruiken van code.

Er is zo veel mogelijk rekening gehouden met uitbreiding, dus

- Character: Bevat gemeenschappelijk eigenschappen bedoeld voor de subklasses ervan afgeleid, zelf bevat het geen methodes, maar de optie is er wel om standaard functies te geven voor elke character

```
public class Character{
    //Standaard waardes voor Characters
    public int[] xPosition, yPosition;
    public ConsoleColor Colour;
    public int Speed;
    public int Length;
    public (int x, int y) StartPosition;
}
```

- Snake: Snake is een subklasse van Character, het bevat de standaard waardes en zijn eigen methodes

```
public class Snake : Character {
    //Constructor voor slang, Wall om te kiezen tussen welke wall hij komt en
kleur omdat we allemaal andere kleuren slang hebben
    public Snake(Wall wall, ConsoleColor colour) {
        Colour = colour;
        Speed = 8;
        Length = 1;
        StartPosition = (wall.Width / 2, wall.Height / 2);
    }
```

- Interfaces: Er is geen gebruik gemaakt van interfaces omdat de implementatie hiervan problemen gaf aangezien de klasses die hiervan gebruik zouden moeten maken voor hun methodes andere parameters nodig zouden hebben.
- Apple: Apple is een subklasse van Character, naast de standaard waardes heeft het een extra eigenschap in de vorm van een Lijst.

```
public class Apple : Character {
    //Lijst van appels zodat ze makkelijk bijgehouden kunnen worden
    public static List<Apple> Apples = [];

    //Een constructor voor een appel, enige parameter is kleur
    public Apple(Wall wall, ConsoleColor colour){
        Colour = colour;
        Speed = 0;
        Length = 1;
        StartPosition = (new Random().Next(1, wall.Width - 1), new
Random().Next(1, wall.Height - 1));
    }
```

- Wall: Deze klasse bevat alleen een simpele methode om eenmalig iets te tekenen, er wordt een object van gemaakt om te kunnen refereren naar de waardes daarvan.

```
public Wall(int width,int height) {
    Height = height + 2;
    Width = width + 2;
}
```

- Score: De score klasse bedoeld om score bij te houden combineert waardes vanuit andere klasses om resultaten te kunnen weergeven

```
public class Score{
     public int CurrentScore = 0;

}
```

- Game: Deze klasse bevat op het moment alleen methodes, wel is er een mogelijkheid om hier een constructor voor de maken die alle gegevens voor het spel opslaat.

```
public class Game {
    public bool gameOver = false;
    Stopwatch timer = new();

}
```

5 Behavioral Pattern

Behavioral patterns oftewel gedragspatronen zijn ontwerppatronen die zicht bezighouden hoe objecten met elkaar interageren. Deze patronen gaan over de manier waarop klassen en objecten met elkaar samenwerken. En hoe ze taken op elkaar aftstemmen. Verder helpen gedragspatronen bij het definieren van algoritmen, comminucatieptronen tussen objecten. Hierdoor wordt de flexibiliteit van systemen verbeterd.

Snake

De beweging: De slang reageert op de invoer die gegeven wordt door de gebruiker, bijvoorbeeld: de richting van de slang. Op het moment dat de slang een appel eet, versnelt de snelheid van de slang in het speelveld.

Groei: Wanneer de slang een appel eet, neemt de lengte van de slang toe (de staart). Dit houdt in dat bij elke stap die de slang neemt, dat het uiteinde van de staart verplaatst met de slang (waar de kop van de slang eerder was).

Zelfbotsing: Als de slang tegen zijn eigen staart botst, resulteert dit in het einde van het spel. Wanneer de slang tegen zijn eigen lichaam aan botst, wordt binnen de Game klassen het spel beëindigd.

Muurbbotsing: Als de slang tegen de muur of border aanbotst eindigt het spel. Wanneer de slang buiten het speelveld resulteert dit binnen de Game klassen game over.

```
public class Snake : Character {
    //Constructor voor slang, Wall om te kiezen tussen welke wall hij komt en
kleur omdat we allemaal andere kleuren slang hebben
    public Snake(Wall wall, ConsoleColor colour) {
        Colour = colour;
        Speed = 8;
        Length = 1;
        StartPosition = (wall.Width / 2, wall.Height / 2);
    }

    //Tekent de slang in slangenkleur en tekent over zijn spoor
    public void Draw(Snake snake) {
        if(snake.xPosition == null && snake.yPosition == null) {
            Console.SetCursorPosition(StartPosition.x, StartPosition.y);
        }
        if(snake.xPosition != null && snake.yPosition != null){
            Console.SetCursorPosition(snake.xPosition[0], snake.yPosition[0]);
        }
        Console.ForegroundColor = snake.Colour;
        Console.Write("\u2558");
        Console.ResetColor();
```

```
        //Omdat je de console niet elke keer meer cleart moet je dit handmatig
doen door naar de slang zijn staart
        //Te gaan en het handmatig over te kleuren. Vergeet niet de cursor terug
te doen.
        if(snake.xPosition != null && snake.xPosition.Count() > snake.Length){
            (int x, int y) currentPosition = (snake.xPosition[0],
snake.yPosition[0]);
            Console.SetCursorPosition(snake.xPosition[snake.Length],
snake.yPosition[snake.Length]);
            Console.ForegroundColor = ConsoleColor.Black;
            Console.Write("\u2558");
            Console.ResetColor();
            Console.SetCursorPosition(currentPosition.x, currentPosition.y);
        }
    }

    public void Update(Snake snake, Apple apple, Score score, ConsoleKeyInfo
keyInfo){
        int prevX = snake.xPosition[0], prevY = snake.yPosition[0];
        int prev2X, prev2Y;

        // Beweging van het hoofd van de slang
        switch(keyInfo.Key) {
            case ConsoleKey.RightArrow:
                snake.xPosition[0]++;
                Thread.Sleep((1000 / snake.Speed) / 2);
                break;
            case ConsoleKey.LeftArrow:
                snake.xPosition[0]--;
                Thread.Sleep((1000 / snake.Speed) / 2);
                break;
            case ConsoleKey.DownArrow:
                snake.yPosition[0]++;
                Thread.Sleep((1000 / snake.Speed));
                break;
            case ConsoleKey.UpArrow:
                snake.yPosition[0]--;
                Thread.Sleep((1000 / snake.Speed));
                break;
        }

        // Beweging van de staart van de slang
        for(int i = 1; i < snake.Length; i++) {
            prev2X = snake.xPosition[i];
            prev2Y = snake.yPosition[i];
            snake.xPosition[i] = prevX;
            snake.yPosition[i] = prevY;
            prevX = prev2X;
            prevY = prev2Y;
        }

        //Als de slang positie over een appel positie gaat, meer lengte, meer
snelhid, meer punten, nog meer punten hoe sneller je bent!
        if(apple.xPosition == snake.xPosition && apple.yPosition ==
snake.yPosition) {
            snake.Length++;
            snake.Speed++;
            //Gebaseerd op de slang zijn snelheid krijgt hij meer punten
            score.CurrentScore += (10+(10*(snake.Speed/100)));
        }
    }
}
```

Apple

Plaatsing: de appel wordt willekeurig op het speelveld getoont, elke keer nadat de appel opgegeten is door de slang. Hierdoor wordt de score van de speler verhoogd en de snelheid van de slang. De appel kan niet op dezelfde plek toegevoegd worden als waar de appel stond.

```
public class Apple : Character {
    //Lijst van appels zodat ze makkelijk bijgehouden kunnen worden
    public static List<Apple> Apples = [];

    //Een constructor voor een appel, enige parameter is kleur
    public Apple(Wall wall, ConsoleColor colour){
        Colour = colour;
        Speed = 0;
        Length = 1;
        StartPosition = (new Random().Next(1, wall.Width - 1), new
Random().Next(1, wall.Height - 1));
    }

    //Tekenfunctie voor appel, als hij aangeroepen wordt zoekt hij een
willekeurige plek om een appel te tekenen en voeg hem toe aan de appel lijst!
    public void Draw(Wall wall, Apple apple) {
        apple.xPosition[0] = new Random().Next(1, wall.Width - 1);
        apple.yPosition[0] = new Random().Next(1, wall.Height - 1);
        Console.SetCursorPosition(apple.xPosition[0], apple.yPosition[0]);
        Console.ForegroundColor = apple.Colour;
        Console.Write("\u2558");
        Console.ResetColor();
        Apples.Add(apple);
        //Ze de cursor terug naar die waar de slang zou zijn
        //Console.SetCursorPosition(oude positie);
    }

    public void Update(Wall wall, Apple apple) {
        //Als er meer dan 1 appel zou zijn, verwijder de appel uit de appel lijst
        if(Apples.Count() > 1) {
            Apples.Remove(apple);
        }
        //Als er geen appels zijn, roep de Draw() methode aan
        if(Apples.Count() == 0) {
            Draw(wall, apple);
        }
    }
}
```

Wall

Detectie: Als de slang de grens van het speelveld raakt, resulteert dit ook in het einde van het spel. Wanneer de slang de muur aanraak, roept hij de game over methode uit de Game klassen aan. Waardoor het spel afgelopen is.

Aanmaak: Doormiddel van de Draw klassen wordt de border of muren rond om het scherm aangemaakt wat het speelveld aantoont.

```
//Simpele class voor muur tekenen
public class Wall{
    public int Height, Width;

    public Wall(int width,int height) {
        Height = height + 2;
```

```
        Width = width + 2;
    }

    public void Draw() {
        for(int h = 0; h <= Height; h++) {
            for(int w = 0; w <= Width; w++) {
                if((w == 0 || w == Width) && !(h == 0 || h == Height)) {
                    Console.Write("\u2551");
                }
                if(!(w == 0 || w == Width) && (h == 0 || h == Height)) {
                    Console.Write("\u2550");
                }
                if(!(w == 0 || w == Width) && !(h == 0 || h == Height)) {
                    Console.Write(" ");
                }
                if(w == 0 && h == 0) {
                    Console.Write("\u2554");
                }
                if(w == Width && h == 0) {
                    Console.Write("\u2557");
                }
                if(w == 0 && h == Height) {
                    Console.Write("\u255A");
                }
                if(w == Width && h == Height) {
                    Console.Write("\u255D");
                }
            }
            Console.Write("\n");
        }
    }
}
```

Score

Teller: de score gaat met 10 punten omhoog wanneer de slang een appel heeft gegeten. Hij kan maar 1 appel per keer eten, en er wordt elke keer wordt er 1 appel getoond.

Weergaven: De score wordt op het scherm weergeven onder de border of muren van het speelveld. Hierdoor kan de speler zijn score terugvinden. Binnen de Score klassen hebben we display, reset, leaderboard en current scoren klassen en methode aangemaakt waarbij de score wordt bijgehouden.

```
public class Score{
     public int CurrentScore = 0;
     //Dictionary<string, int> Leaderboard = [];
     //Dictionary kan gebruikt worden voor een leaderboard. Leadboard.Add("Naam",
Score);

     public void AddToLeaderboard(Score score){
         //Gebruiker krijgt de mogelijkheid om zijn naam in te vullen met zijn
huidige score.
         //"Vul je naam in op op het leaderboard te komen!"
         //Leaderboard.Add(Console.ReadLine(), score.CurrentScore);
     }

     public void Display(){
         //Insert behaviours for current score and high score
     }
```

```
    public void Reset(){
        //Wanneer de game opnieuw start
        //score.currentScore = 0;
    }
}
```

Time

Tijd: Wanneer de slang beweegt in het speelveld, loopt de stopwatch. De stopwatch stopt met tellen wanneer de game is gepauzeerd.

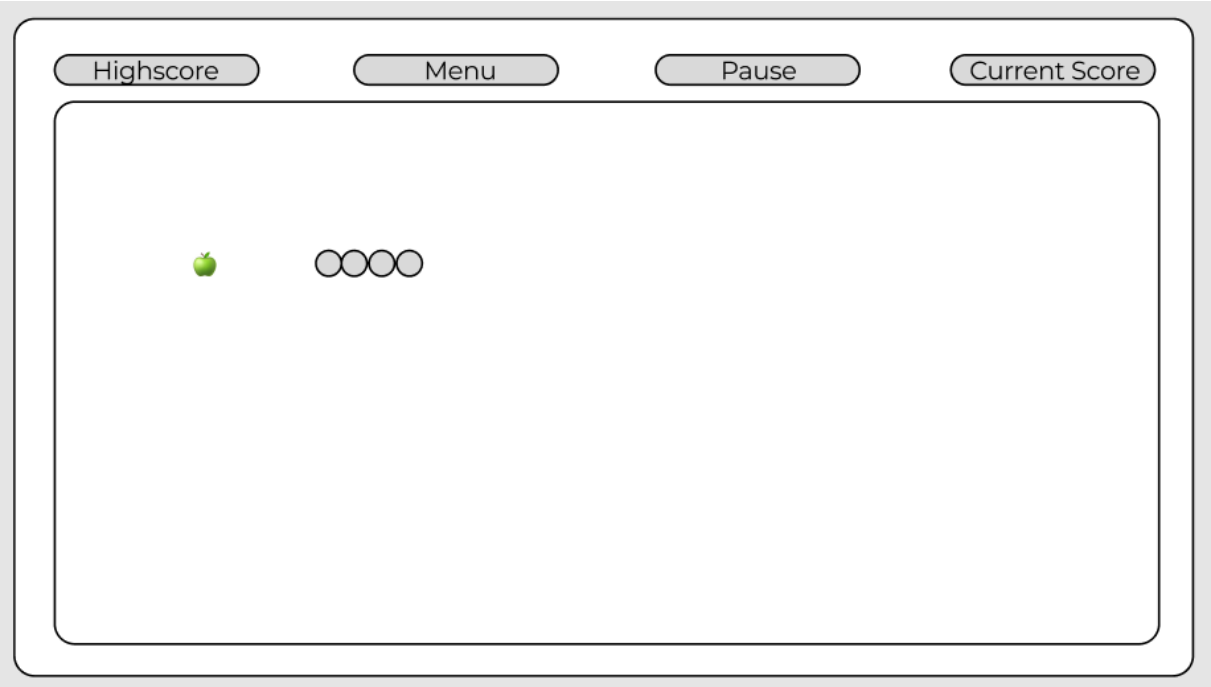
Stopwatch: Wanneer de speler een input invoert begint de timer te lopen. Indien de speler op ‘p’ drukt pauzeert het spel. Wanneer het spel is afgelopen door dat de slang in de muur of in zichzelf gaat, eindigt de timer. De tijd wordt bewaard en weergeven in het scherm na afloop.

```
//Class bevat alles wat over de game status gaat
public class Game {
    public bool gameOver = false;
    Stopwatch timer = new();

    //Wordt aan het begin aangeroepen om het spel te starten
    //Op het moment zorgt het alleen dat ze slang in het midden komt
    public void Start(Snake snake, Score score) {
        //Plaatst de Slang in het midden van het speelveld om te beginnen met het
spel :)
        Console.SetCursorPosition(snake.StartPosition.x, snake.StartPosition.y);
            if(Console.KeyAvailable) {
                timer.Start();
                score.CurrentScore = 0;
                Apple.Apples.Clear();
            }
        }

        public void GameOver(Snake snake, Wall wall, Score score) {
            if (snake.xPosition[0] < 0 || snake.xPosition[0] >= wall.Width ||
snake.yPosition[0] < 0 || snake.yPosition[0] >= wall.Height){
                gameOver = true;
            }

            // Checkt of de slang tegen zichzelf aan botst
            for (int i = 1; i < snake.Length; i++){
                if (snake.xPosition[0] == snake.xPosition[i] &&
snake.yPosition[0] == snake.yPosition[i]){
                    gameOver = true;
                }
            }
            timer.Stop();
            Console.WriteLine($"Score: {score.CurrentScore}");
            Console.WriteLine($"Time: {timer.Elapsed.ToString(@"mm\:ss")}");
        }

    public void Pause(ConsoleKeyInfo keyInfo) {
        if (Console.KeyAvailable) {
            keyInfo = Console.ReadKey(true); // true om te voorkomen dat de toets
wordt weergegeven in de console
            if (keyInfo.Key == ConsoleKey.P){
                Console.WriteLine("\nSpel gepauzeerd. Druk op 'P' om te
hervatten.");
                bool paused = true;
```

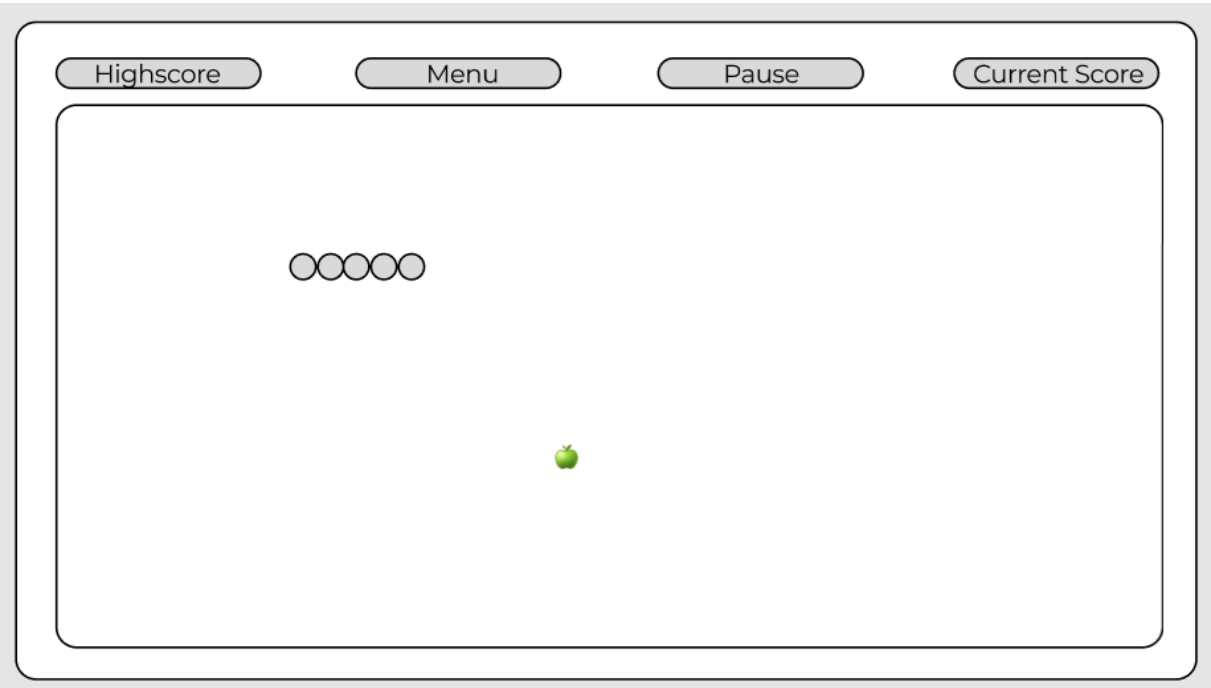
```
                while(paused) {
                    if(Console.KeyAvailable) {
                        ConsoleKeyInfo keyPress = Console.ReadKey(true);
                        if(keyPress.Key == ConsoleKey.P) {
                            paused = false;
                        }
                    }
                }
            }
        }
    }
}
```

5 Wireframes

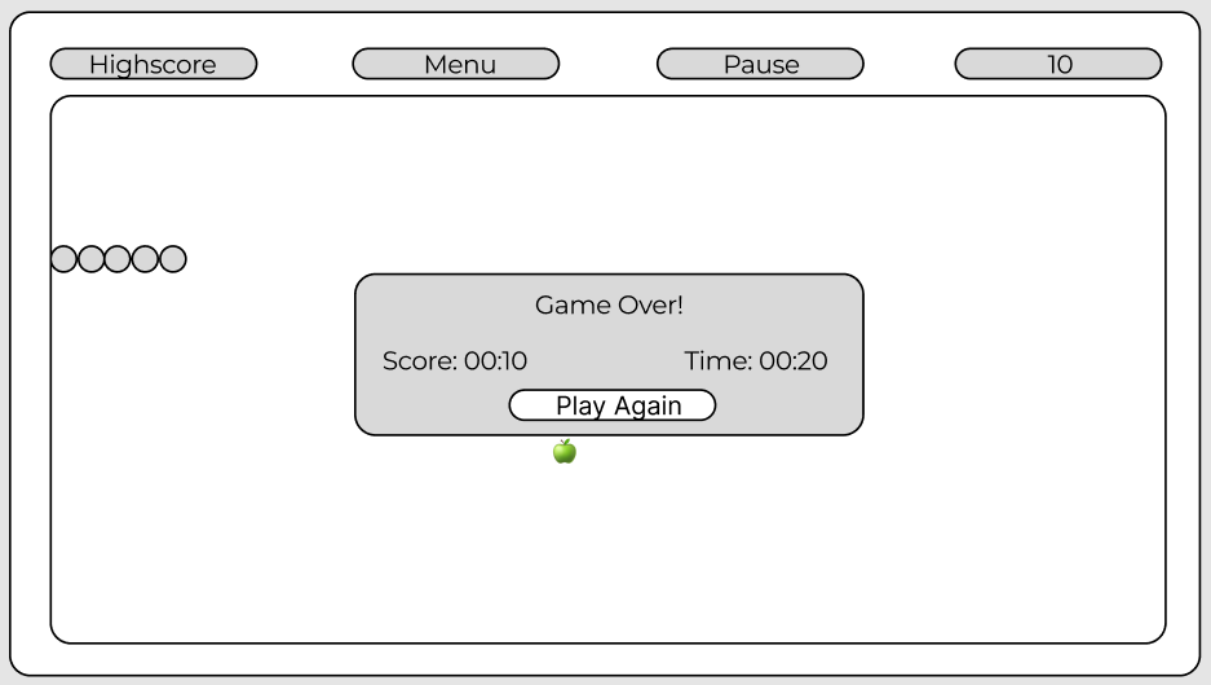
Dit is het beginsituatie van de game.



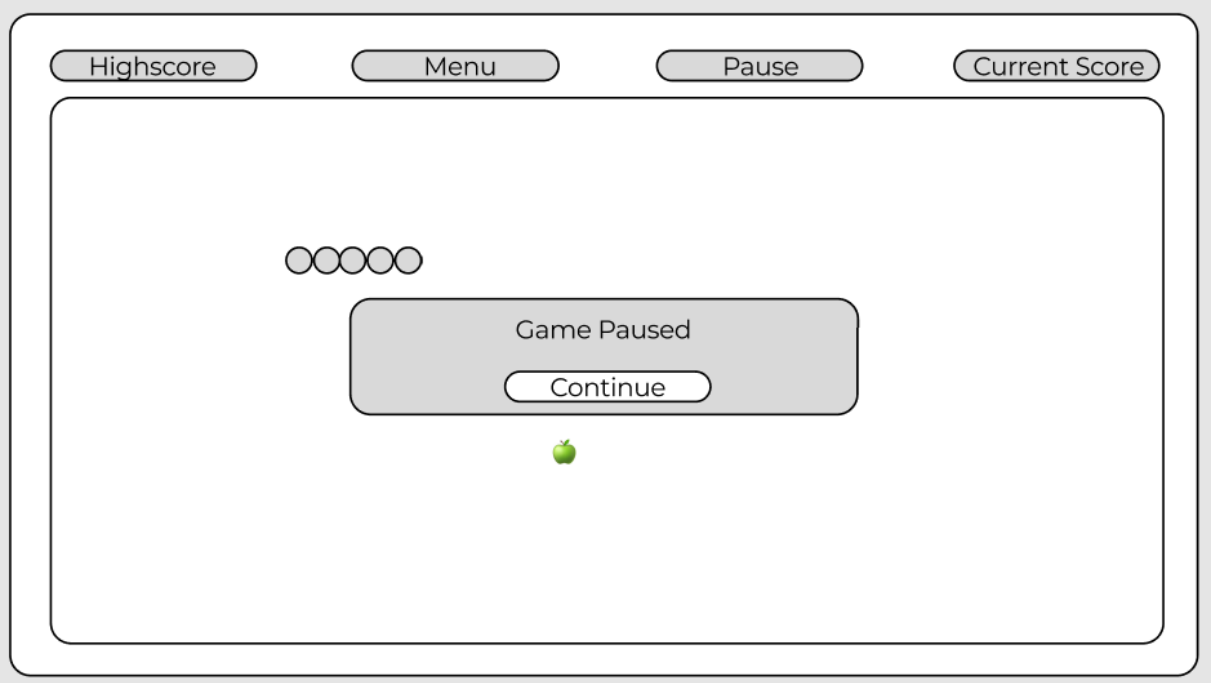
In deze frame is te zien dat de slang langer is geworden na het eten van de groene appel.



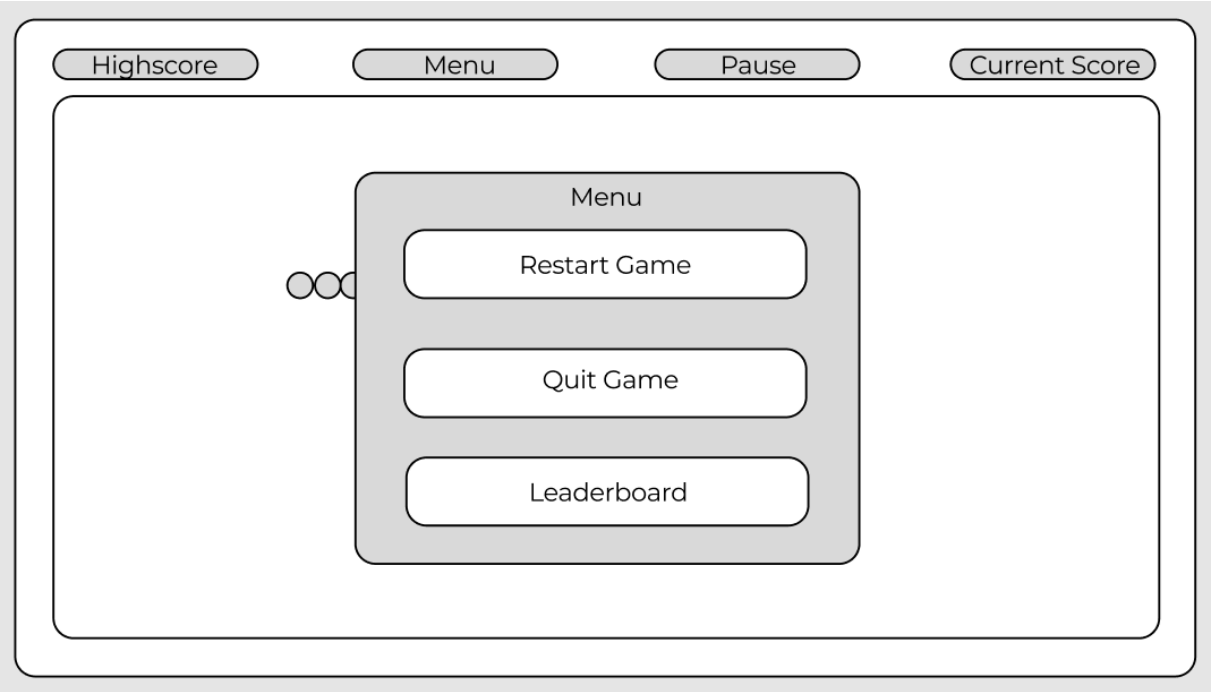
Het is Game Over! Wanneer de slang een muur raakt met zijn kop.

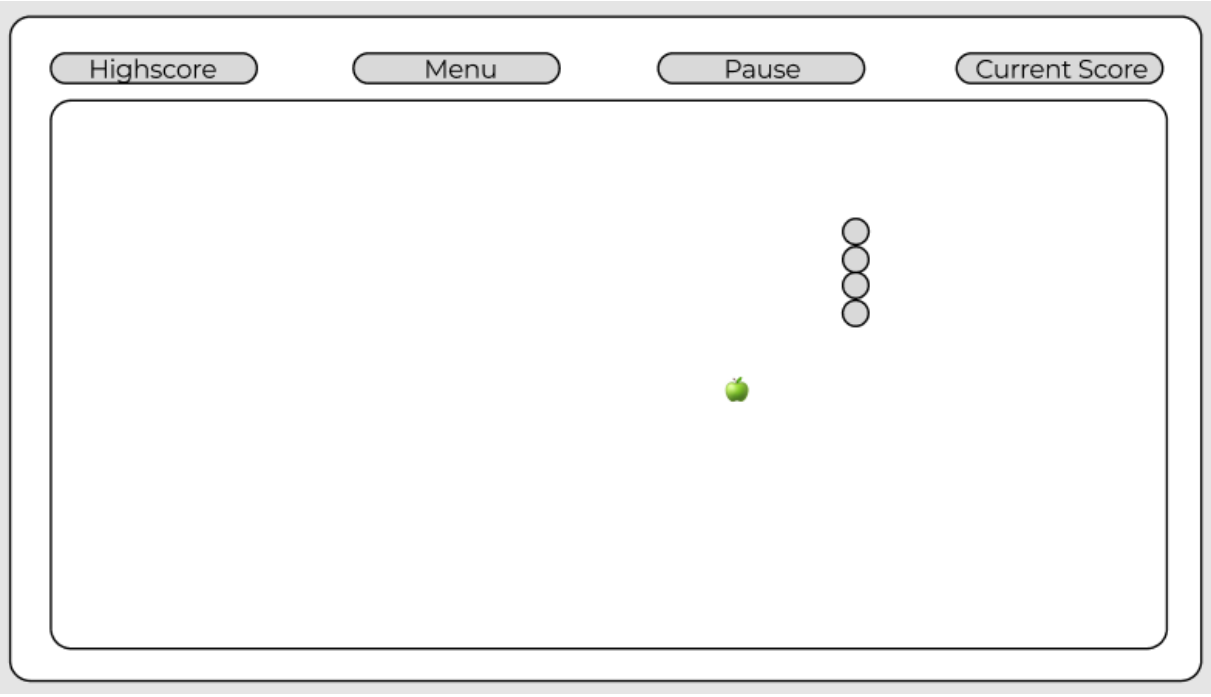


Je kan de game pauzeren door op Pause te klikken.

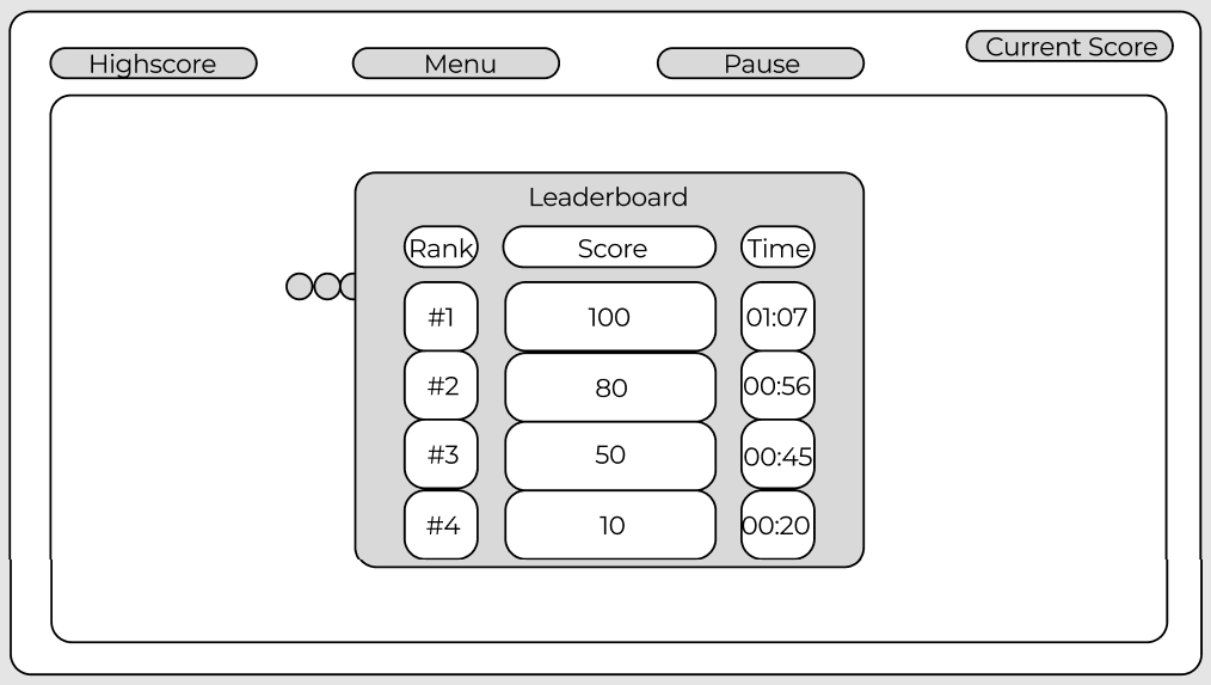


Door op Menu te klikken verschijnt het menu. Je kan ervoor kiezen om de game te restarten, stoppen of de leaderboard te bekijken. Wanneer je op Quit Game klikt, dan begint er een nieuwe game.





Hier is de Leaderboard te zien.



6 Reflectie

6.1 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

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

Sprint 2

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

Sprint 3

Tijdens deze sprint vonden we het lastig om te werken aan het project, want we richtten ons meer op andere projecten.

Sprint 4

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

6.2 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.

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