Software Design Documentation

Product Name	Pentomino
Date Updated	Juni
Written By	Barry van den Berg

Introduction

Dit document biedt een uitgebreide ontwerpoverzicht van het Tetris-spel met een twist. Het doel is om de architectuur, componenten en interacties gedetailleerd te beschrijven. Dit document dient als handleiding voor ontwikkelaars, testers en onderhouders van het systeem.

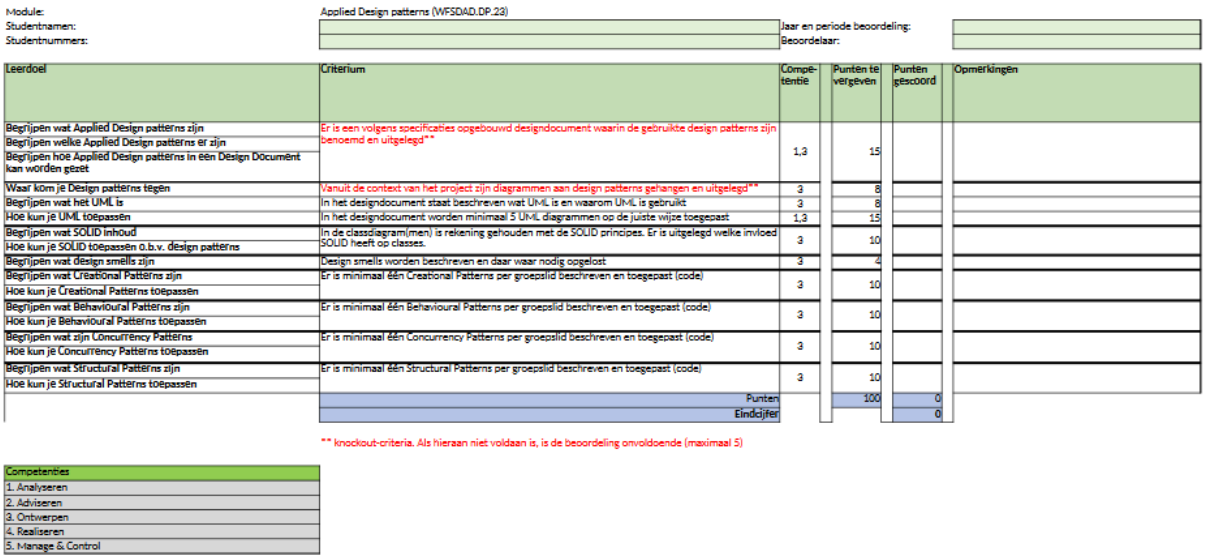
1.2 Scope

Dit document behandelt de ontwerpaspecten van het Tetris-spel, inclusief het spelrooster, de blokken, het statusbeheer, het geluidsbeheer en de gebruikersinterfacecomponenten.

1.3 Definitions and Acronyms

- **Grid**: Het speelgebied waar blokken worden geplaatst.
- **Cell**: Een individuele eenheid binnen het rooster.
- **State**: Vertegenwoordigt de huidige status van het spel, inclusief actieve blokken.
- **ScoreBoard**: Verte
- **Observer Pattern**: Een ontwerpprincipe dat wordt gebruikt om waarnemers te informeren over wijzigingen in het onderwerp.

Beoordelings Criteria



Module:	Applied Design patterns (WFSDAD.DP.23)		
Studentnamen:		Jaar en periode beoordeling:	
Studentnummers:		Beoordelaar:	

Leerdoel	Criterium	Compe-tentie	Punten te vergeven	Punten gescoord	Opmerkingen
Begrijpen wat Applied Design patterns zijn Begrijpen welke Applied Design patterns er zijn Begrijpen hoe Applied Design patterns in een Design Document kan worden gezet	Er is een volgens specificaties opgebouwd designdocument waarin de gebruikte design patterns zijn benoemd en uitgelegd**	1,3	15		
Waar kom je Design patterns tegen	Vanuit de context van het project zijn diagrammen aan design patterns gehangen en uitgelegd**	3	8		
Begrijpen wat het UML is	In het designdocument staat beschreven wat UML is en waarom UML is gebruikt	3	8		
Hoe kun je UML toepassen	In het designdocument worden minimaal 5 UML diagrammen op de juiste wijze toegepast	1,3	15		
Begrijpen wat SOLID inhoud Hoe kun je SOLID toepassen o.b.v. design patterns	In de classdiagram(men) is rekening gehouden met de SOLID principes. Er is uitgelegd welke invloed SOLID heeft op classes.	3	10		
Begrijpen wat design smells zijn	Design smells worden beschreven en daar waar nodig opgelost	3	4		
Begrijpen wat Creational Patterns zijn Hoe kun je Creational Patterns toepassen	Er is minimaal één Creational Patterns per groepslid beschreven en toegepast (code)	3	10		
Begrijpen wat Behavioural Patterns zijn Hoe kun je Behavioural Patterns toepassen	Er is minimaal één Behavioural Patterns per groepslid beschreven en toegepast (code)	3	10		
Begrijpen wat zijn Concurrency Patterns Hoe kun je Concurrency Patterns toepassen	Er is minimaal één Concurrency Patterns per groepslid beschreven en toegepast (code)	3	10		
Begrijpen wat Structural Patterns zijn Hoe kun je Structural Patterns toepassen	Er is minimaal één Structural Patterns per groepslid beschreven en toegepast (code)	3	10		
	Punten		100	0	
	Eindcijfer			0	

** knockout-criteria. Als hieraan niet voldaan is, is de beoordeling onvoldoende (maximaal 5)

Competenties
1. Analyseren
2. Adviseren
3. Ontwerpen
4. Realiseren
5. Manage & Control

System Overview

Het Tetris-spel is een desktop-applicatie die spelers uitdaagt om blokken in verschillende vormen te manipuleren en lijnen te vullen. Het spel bevat unieke functies zoals zowel 4 cellige blokken maar ook met 5 cellen. De applicatie is gebouwd met C# en .NET voor de backend en WPF voor de frontend.

Design Considerations

- **Gebruiksvriendelijke interface**: Het spel moet intuïtief en gemakkelijk te gebruiken zijn.
- **Schaalbaarheid**: Het systeem moet flexibel zijn voor toekomstige uitbreidingen.
- **Beveiliging**: Gebruikersinformatie moet veilig worden opgeslagen indien dit word opgeslagen
- **Compatibiliteit**: De applicatie moet compatibel zijn met verschillende versies van Windows.

Design Specifications

Requirement	Description
R1	Er moet een scherm komen die de grid laat zien voor de game.
R2	De gebruiker moet het spel kunnen starten.
R3	Implementeren van Grid klassen om het scherm de controleren
R4	De gebruiker moet blokken naar links en rechts kunnen bewegen
R5	Blokken moeten gedraaid kunnen worden
R6	Blokken moeten sneller bewogen worden (soft drop & hard drop)
R7	Blokken moeten geplaatst worden wanneer deze de bodem raken bij drop
R8	Blokken moeten geplaatst worden als deze andere blokken raken bij drop
R9	Rijen moeten geleegd worden wanneer deze vol zijn.
R10	Andere Rijen moeten dalen wanneer lege rijen verwijdered worden.
R11	Blokken moeten verschillende kleuren hebben op basis van blok ID

R12	Er moet een game loop komen waarin bepaalde dingen worden uitgevoerd
R13	Blokken moeten automatisch dalen elke game loop
R14	Er moet een score class komen die update wanneer er een lijn af is
R15	De game moet eindigen wanneer een blok geplaatst word op een te hoge rij.
R16	De game moet geluiden afspelen op basis van acties
R17	Er moet een game end scherm komen waarbij je de score kan zien.
R18	De code moet goed gedocumenteert worden met duidelijk comments.

Detailed Design

Overzicht

Dit gedeelte biedt een gedetailleerd ontwerp van het Pentomino Tetris-spel, inclusief de patterns. Deze details zijn essentieel voor ontwikkelaars, testers en onderhouders om een duidelijk begrip van het systeem te krijgen.

Componenten

1. Grid Component

- **Beschrijving**: Vertegenwoordigt het speelrooster waarin de blokken worden geplaatst.
- **Belangrijkste Klassen**:

    - `Grid`: Beheert het rooster en de plaatsing van blokken.
    - `Cell`: Vertegenwoordigt een individuele cel in het rooster.

2. State Component

- **Beschrijving**: Beheert de huidige status van het spel, inclusief actieve blokken en hun bewegingen.
- **Belangrijkste Klassen**:
    - `State`: Beheert de huidige status van het spel en de actieve blokken.
    - `Block`: Vertegenwoordigt een blok en zijn eigenschappen.

3. ScoreBoard Component

- **Beschrijving**: Houdt de score van de speler bij en geeft deze weer.
- **Belangrijkste Klassen**:
    - `ScoreBoard`: Beheert de score en geeft deze weer in de gebruikersinterface.

4. SoundManager Component

- **Beschrijving**: Beheert de geluidseffecten van het spel.
- **Belangrijkste Klassen**:
    - `SoundManager`: Speelt geluidseffecten af bij specifieke gebeurtenissen in het spel.

5. User Interface (UI) Component

- **Beschrijving**: Beheert de gebruikersinterface van het spel.
- **Belangrijkste Klassen**:
    - `MainWindow`: Beheert de verschillende schermen en gebruikersinteracties.
    - `GameCanvas`: Het hoofdmenu van het spel.
    - `Grid`: De locaties van de cellen die in de canvas zitten.

Design Patterns

1. Observer Pattern

- **Toepassing**: Gebruikt om de Grid- en State-componenten te observeren voor wijzigingen en deze wijzigingen door te geven aan de SoundManager, MainWindow, ScoreBoard-componenten, etz.
- **Belangrijkste Klassen**:
    - `IClearObserver`: Interface voor het observeren van rijen die worden gewist.
    - `IBounceObserver`: Interface voor het observeren van botsingen.
    - `IRotateObserver`: Interface voor het observeren van rotaties.
    - `IHardDropObserver`: Interface voor het observeren van harde drops.

- `ISoftDropObserver`: Interface voor het observeren van zachte drops.
- `ILevelObserver`: Interface voor het observeren van spel level (niveau / difficulty)
- `ISoftDropObserver`: Interface voor het observeren van score aanpassing.
- `IMoveObserver`: Interface voor het observeren van bewegingen.

```
internal interface IClearSubject
{
  void AttachClearObserver(IClearObserver observer);
  void DetachClearObserver(IClearObserver observer);
  void ClearNotify(int dropAmount);
}

internal interface IClearObserver
{
  void ClearUpdate(int clearedRows);
}

internal class Grid : IClearSubject
{
  //
  //
  //
  private readonly List<IClearObserver> = new()

  public void AttachClearObserver(IClearObserver observer)
  {
     clearObservers.Add(observer);
  }
  public void DetachClearObserver(IClearObserver observer)
  {
     clearObservers.Remove(observer);
  }
  public void ClearNotify(int clearedRows)
  {
     foreach (var observer in clearObservers)
     {
        observer.Update(clearedRows);
     }
  }
  //
  //
  //
}
```

2. Singleton Pattern

- **Toepassing**: Gebruikt om ervoor te zorgen dat er slechts één instantie van de ScoreBoard en SoundManager klassen is.
- **Belangrijkste Klassen**:
    - `ScoreBoard`: Singleton klasse die de score bijhoudt.
    - `SoundManager`: Singleton klasse die geluidseffecten beheert.

3. Composite Pattern

- **Toepassing**: Gebruikt voor de mogelijkheid van samenstellen van complexe UI-elementen en de relatie tussen `Grid` en `Cell`.
- **Belangrijkste Klassen**:
    - `Grid`: Composiet die meerdere `Cell` objecten bevat.
    - `Cell`: Bladklasse die een individuele cel in het rooster vertegenwoordigt.
    - `Image`: De class die de cell heeft voor display.
    - `Canvas`: Het scherm waarop de images getekent worden.

```
internal interface IGridComponent
{
  void Add(IGridComponent child)
  void Remove();
  IEnumarable<IGridComponent> GetChildren();
  void Operation(int value);
  void Draw();
}

internal class Grid : IGridComponent
{
  private readonly Canvas gameCanvas;
  private readonly cellSize = 25;
  //
  //
  //
  public void Add(Cell cell)
  {
    Image image = new()
    {
      Width = cellsize;
      Height = cellSize;
    }
    Canvas.SetTop(imageControl, (cell.Row - 2) * cellSize + 10);
```

```
    Canvas.SetLeft(imageControl, cell.Column * cellSize);
    gameCanvas.Children.Add(imageControl);
    cell.Icon = imageControl;
    cells[cell.Row, cell.Column] = cell;
  }

  public void Remove()
  {
    gameCanvas.Children.Clear();
  }

  public void IEnumarable<IGridComponent> GetChildren()
  {
    for (int row = 0; row < Rows; row++)
    {
      for (int column = 0; column < Columns; column++)
      {
        yield return cells[row, column];
      }
    }
  }

  public void Operation(int value = 0)
  {
    for (int row = 0; row < Rows; row++)
    {
      for (int column = 0; column < Columns; column++)
      {
        Add(new Cell(row, column));
      }
    }
  }

  public void Draw()
  {
    foreach (Cell cell in GetChildren().Cast<Cell>())
    {
      cell.Draw();
    }
  }
  //
  //
  //
}
```

4. Factory Pattern

- **Toepassing**: Gebruikt om blokken (Tetromino's en Pentomino's) te maken via een abstracte fabrieksinterface.
- **Belangrijkste Klassen**:
    - `IBlockFactory`: Interface voor het maken van blokken.
    - `TetrominoFactory`: Concreet fabrieksklasse voor het maken van Tetromino's.
    - `PentominoFactory`: Concreet fabrieksklasse voor het maken van Pentomino's.

```
internal interface IBlockFactory
{
    Block CreateBlock();
}

internal class TetrominoFactory : IBlockFactory
{
    private readonly Block[] tetrominos = new Block[]
    {
        new ITetromino(),
        new JTetromino(),
        new LTetromino(),
        new OTetromino(),
        new STetromino(),
        new TTetromino(),
        new ZTetromino(),
    };

    private readonly Random random = new();

    public Block CreateBlock()
    {
        return tetrominos[random.Next(tetrominos.Length)];
    }
}

internal class Queue
{
  //
  //
  //
  private Block RandomBlock()
  {
    IBlockFactory blockFactory;
    //
```

```
    if (random.Next(100) < pentominoWeight)
    {
        blockFactory = pentominoFactory;
    }
    else
    {
        blockFactory = tetrominoFactory;
    }
    return blockFactory.CreateBlock();
  }
  //
  //
  //
}
```

5. Memento Pattern

- **Toepassing**: Gebruikt om de toestand van een blok op te slaan (bijvoorbeeld een heldere blok die teruggeroepen kan worden).
- **Belangrijkste Klassen**:
    - `State`: Klasse die de toestand van een blok opslaat.
    - `Block`: Klasse die zijn toestand kan opslaan in en herstellen vanuit een `State`.

6. Thread Pool Pattern

- **Toepassing**: Gebruikt voor het beheren van gelijktijdige taken zoals het afspelen van geluiden.
- **Belangrijkste Klassen**:
    - `ThreadManager`: Klasse die een pool van threads beheert voor het uitvoeren van taken.
    - `SoundManager`: Klasse die taken aan de `ThreadManager` toevoegt voor het afspelen van geluidseffecten.
    - `ScoreBoard`: Klasse die taken aan de `ThreadManager` toevoegd voor het bereken van nieuwe scores
    - `MainWindow`: Klasse die taken aan de `ThreadManager` toevoegd voor het runnen van een async loop.

```
internal class ThreadManager : IDisposable
{
    private readonly BlockingCollection<Action> taskQueue = new BlockingCollection<Action>();
    private readonly Thread[] workers;

    public ThreadManager(int workerCount)
    {
```

```
        workers = new Thread[workerCount];
        for (int i = 0; i < workerCount; i++)
        {
            workers[i] = new Thread(Work);
            workers[i].Start();
        }
    }

    public void QueueTask(Action task)
    {
        taskQueue.Add(task);
    }

    public void Work()
    {
        foreach (var task in taskQueue.GetConsumingEnumerable())
        {
            try
            {
                task();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }

    public void Dispose()
    {
        taskQueue.CompleteAdding();
        foreach (var worker in workers)
        {
            worker.Join();
        }
    }
}

internal class ScoreBoard
{
  private readonly ThreadManager threadpool;

  public ScoreBoard(ThreadManager threadpool)
  {
```

```
    this.threadpool = threadpool;
  }

  public void Update(int clearedRows)
  {
    threadpool.QueueTask(() =>
    {
      // Run calculation
    });
}
```

Implementation Plan

Task	Assigned To	Start Date	End Date	Status
Frontend Development	Barry van den Berg	05/APR/2024	01/JUN/2024	Finished
Backend Development	Barry van den Berg	01/JUN/2024	14/JUL/2024	Finished

Maintenance Plan

Het onderhoudsplan beschrijft de aanpak voor het bijhouden en verbeteren van het Pentomino Tetris-spel na de initiële release. Dit omvat regelmatige updates, bugfixes, functieverbeteringen en ondersteuning voor gebruikers. Het plan is ontworpen om de kwaliteit en betrouwbaarheid van de software waar te maken.

1. Regelmatige Updates

- **Frequentie**: Maandelijks of vaker indien nodig.
- **Inhoud**: Kleine verbeteringen, prestatie-optimalisaties en het bijwerken van afhankelijkheden.
- **Verantwoordelijkheid**: Het ontwikkelteam voert deze updates uit en zorgt voor compatibiliteit met bestaande functies.

2. Bugfixes

- **Prioritering**: Kritieke bugs die de werking van het spel ernstig beïnvloeden krijgen prioritei

3. Functieverbeteringen

- **Gebruikersfeedback**: Nieuwe functies en verbeteringen worden gepland op basis van nodig heden.
- **Planning**: Functieverbeteringen worden ingepland tijdens reguliere ontwikkeling cycle.
- **Implementatie**: Implementeert nieuwe functies en zorgt ervoor dat correct integreren met bestaande functies

Conclusion

Dit softwareontwerpdocument biedt een grondig overzicht van het ontwerp van de Tetris-toepassing, met een focus op de toepassing van verschillende ontwerppatronen om specifieke ontwerpproblemen op te lossen.

Creational Patterns

Het Factory-patroon is toegepast voor het maken van blokken (IBlockFactory met PentominoFactory en TetrominoFactory). Dit patroon is gekozen om de creatie van verschillende typen blokken te abstraheren en te centraliseren, waardoor de flexibiliteit en uitbreidbaarheid van het blokmakingsproces worden vergroot.

Behavioral Patterns

De Memento-patroon is gebruikt voor het beheren van de huidige status van het spelblok. Dit patroon maakt het mogelijk om de staat van een object om te wisselen met een ander.

Concurrency Patterns

Het ThreadPool-patroon is geïmplementeerd om gelijktijdige taken te beheren, zoals het afspelen van geluidseffecten. Dit patroon verbetert de prestaties en schaalbaarheid door taken parallel uit te voeren op een threadpool, wat essentieel is voor een responsieve gebruikerservaring.

Structural Patterns

Het Composite-patroon is toegepast voor de Grid naar Cell-relatie. Hiermee kunnen cellen worden behandeld als individuele entiteiten of als deel van een groter geheel (de grid), wat de hiërarchische structuur van het speelveld vereenvoudigt en uniforme operaties mogelijk maakt op individuele cellen en de grid als geheel.

Elk van deze ontwerppatronen draagt bij aan de modulariteit, flexibiliteit en herbruikbaarheid van de codebase, en is zorgvuldig gekozen om specifieke ontwerpuitdagingen aan te pakken binnen de context van het Tetris-spel.

Signatures:

Project Manager: Barry van den Berg