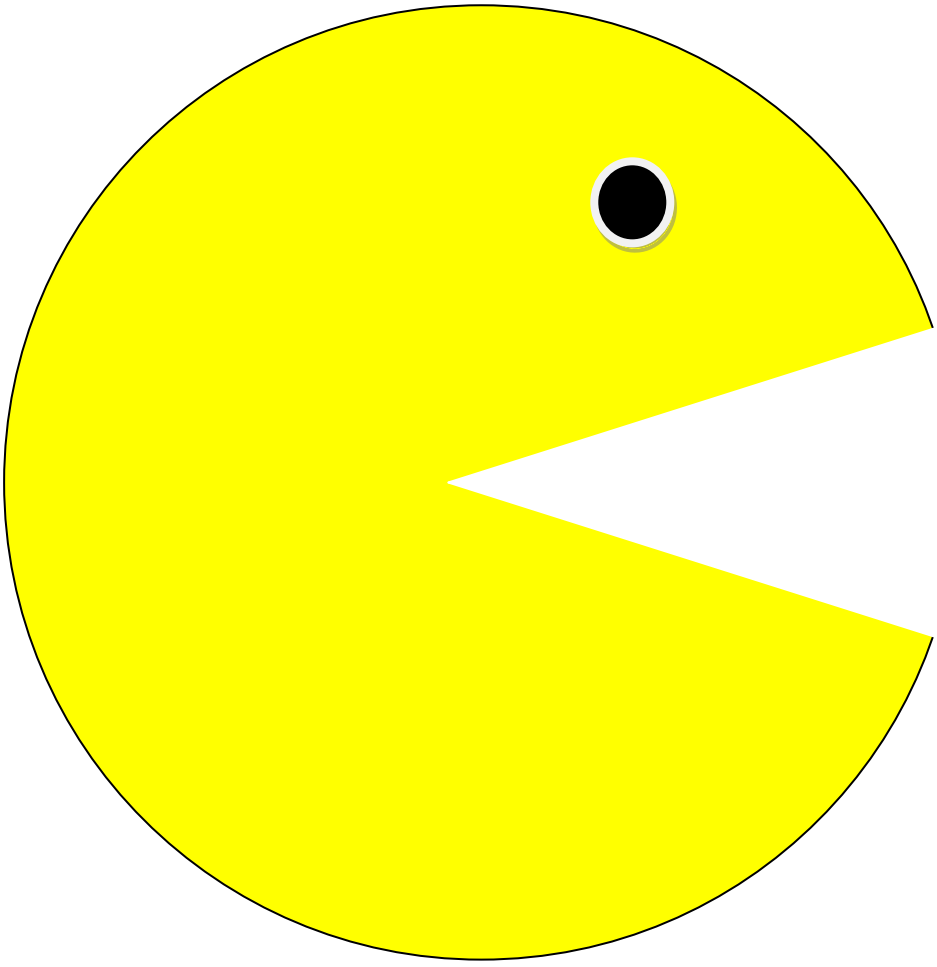
DV1435 Detailed Design

Thomas Sievert, Martin Säll, Lars Woxberg ,Kim Restad &Fredrik Johannesson



thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Contents

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Project description

Pacman::Reloaded is a 3D rendition of the timeless classic Pacman.

The game starts with a title screen, where there are four options: Play the game, view the highscore, view the credits, and quit the game. The game has a set number of stages for Pacman to go through. When he has finished them all, he is sent back to the first stage, on a slightly harder difficulty. Thus, the game goes on indefinitely, or until Pacman dies. When the game is over, the player might be registered to the highscore list. Pacman::Reloaded is developed on Windows for Windows PCs with Direct3D10 compatibility.

Each level has a certain amount of pellets spread out, and the goal is for Pacman to eat all of them while at the same time avoiding the dreaded ghosts, chasing him around. The layout of the level is tile-based and maze-like, and the density of pellets is high. Wherever there isn't a wall tile, there will initially reside a pellet. The only exception to this is the ghosts' home.

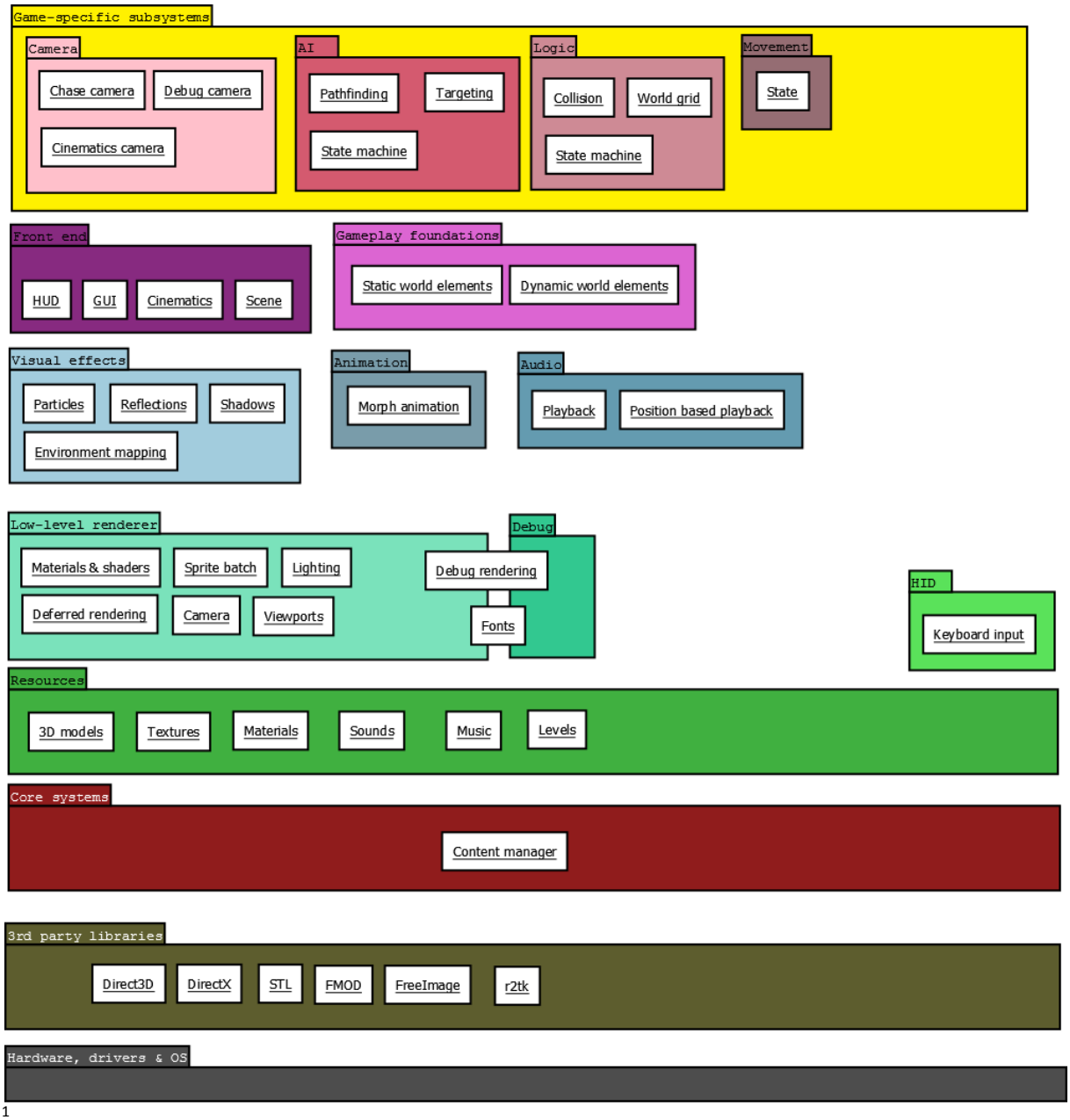
Pacman moves around in a simple manner: in one of four directions. He can turn corners, or turn around on the spot, but he can never stand still. The same rules apply to the ghosts. Whenever Pacman finds himself on a tile where there is a pellet, he eats it, and gains some points. If, however, he finds himself on the same tile as a ghost, he dies.

There are a few special pellets, called "power pellets" that make Pacman invincible and able to eat his enemies for a limited period of time. There may also appear a fruit occasionally, which gives Pacman a large amount of points. If he has not eaten all the pellets within a certain period of time, Pacman dies. The game goes on until he has died three times. If the player has managed to collect a high enough score, he can enter his name on the highscore board.

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Architecture overview

The following diagram describes the functionality in the system using a top-down style approach. The further up along the diagram one climbs, one will find more specifically game related components.

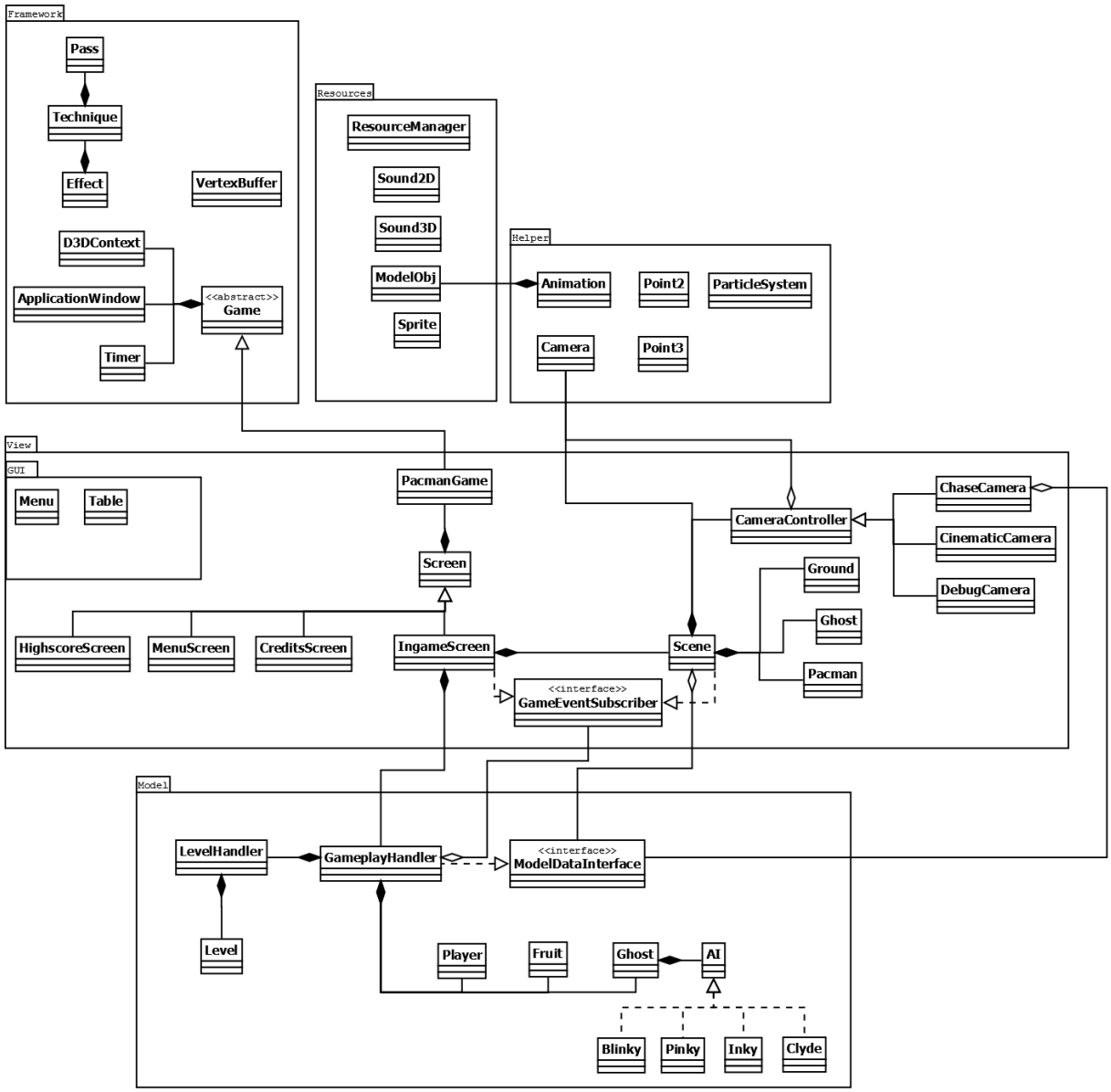


1

[1] ArchitectureOverviewDiagram.png

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Description of components



2

The game is designed very much with a Model/View setup in mind. As can be seen in the diagram above (*ClassOverviewDiagram.png*), the Model package is completely independent, only exposing an interface for the data and the events relevant to the view. The View package also functions as a controller, updating the Model from the Ingame screen component.

The Helper, Framework and Resource packages contain the "basics" and are as such accessed by both the View and the Model, when need arises. We also regard them as less relevant to explain in full detail, since they don't describe the game mechanics as much as they are a foundation.

[2] ClassOverviewDiagram.png

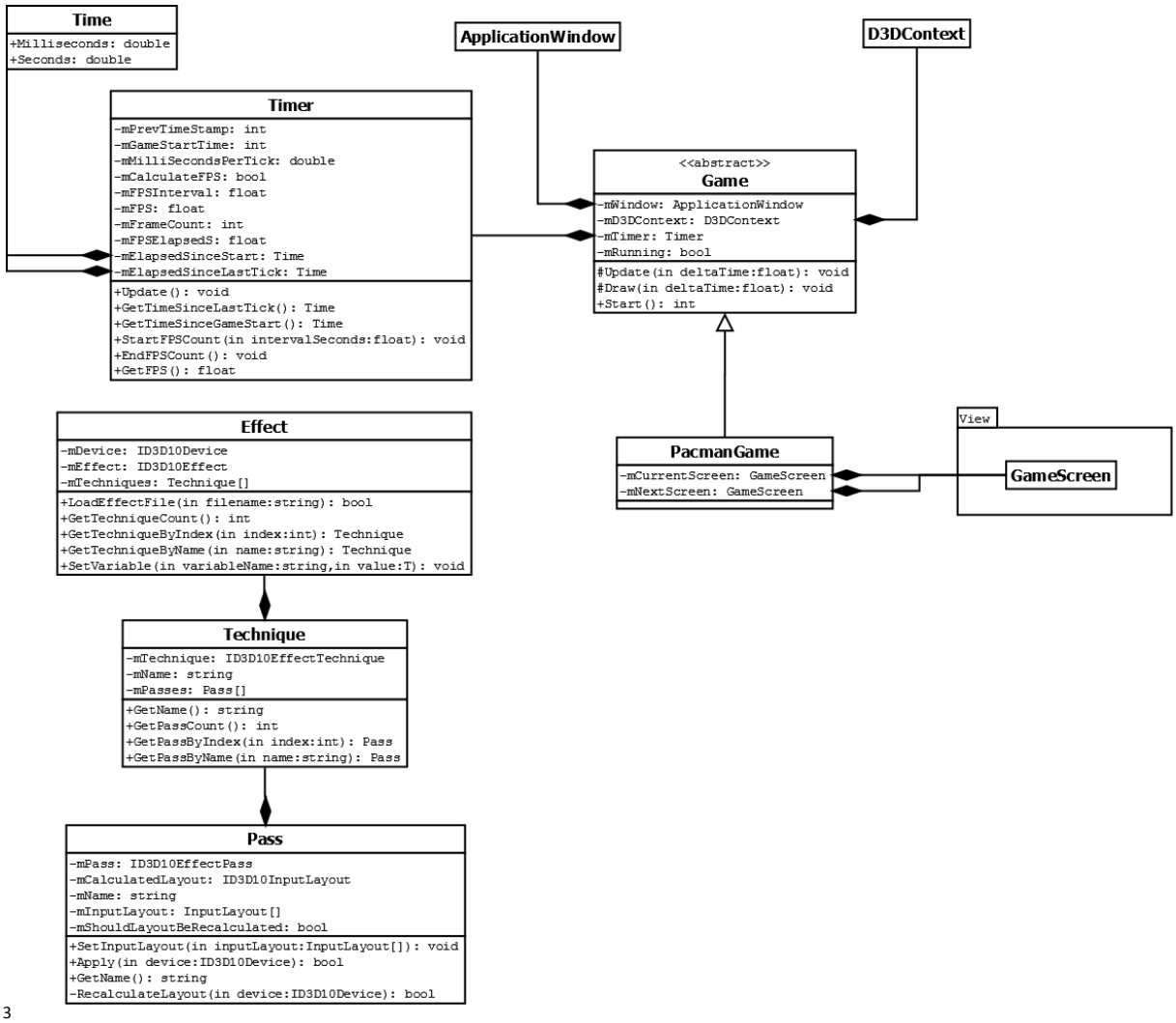
thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Framework

Handles the basic setup of the game, such as creating a window, initializing Direct3D, and provides an abstraction to their services.

- Game
  Wraps around the game loop. Is meant to be extended by the main game class.
- D3DContext
  Abstracts Direct3D functionality.
- ApplicationWindow
  Abstracts Win32 functionality for handling windows.
- Timer
  A class for measuring time with high precision.
- VertexBuffer
  An abstraction for the data structure used to send information to the GPU.
- Effect
  An abstraction to handle effect (.fx) files, i.e. shaders.
    - Pass
      A subcomponent to the Technique class. One Technique may consist of many passes.
    - Technique
      A subcomponent to the Effect class. One Effect may consist of many techniques.

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com



3

The Framework package consists of components abstracting Win32 and Direct3D functionality. These make up the very foundation upon which the game is run. Much of it is boilerplate work that has very little impact on the actual design.

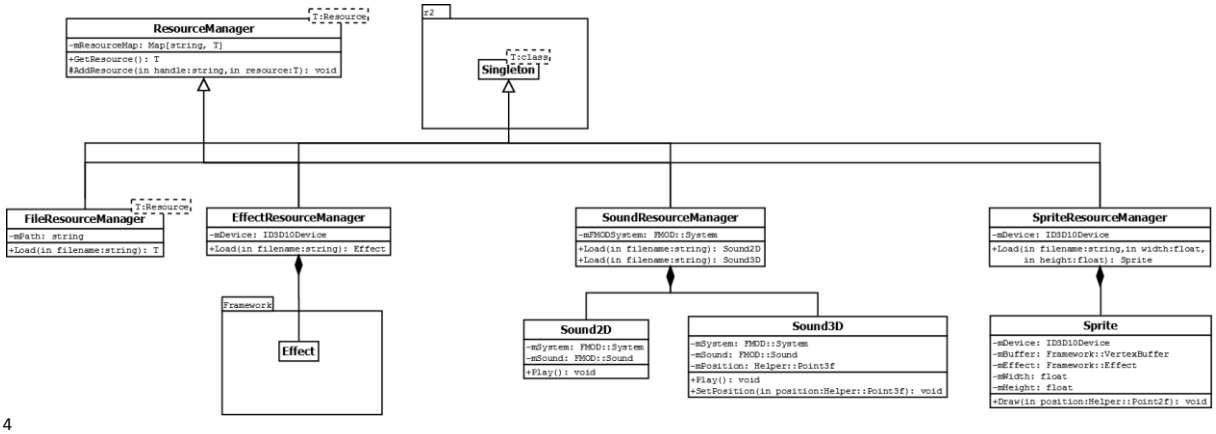
[3] FrameworkClassDiagram.png

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Resources

Manages all resources we do not wish to load twice.

- ResourceManager
  Holds references to resources of a specific type. This is a templated singleton class. Thanks to this, resources are easily available in any part of the project, should they be needed.
- Sound2D
  Plays a sound.
- Sound3D
  Plays a sound from a certain position in a 3D scene.
- ModelObj
  A 3D model loaded from a model file.
- Sprite
  Manages a 2D HUD image.



4

The resources package contains all components concerning the game's various resources. A Singleton pattern is used for the various Resource Managers to make the resources easily accessible from all parts of the system. The Singleton base class from which the managers are derived comes from an open source third party library called r2tk.

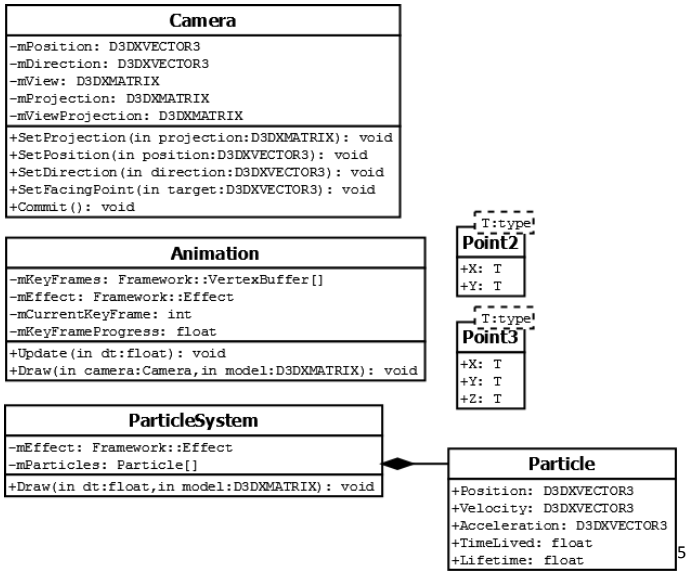
[4] ResourceClassDiagram.png

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Helper

Classes with generic use, needed throughout the system.

- Animation
  A collection of 3D models, using morph animation to interpolate between key frames.
- ParticleSystem
  A system of particles used to create various effects.
- Camera
  A camera used to look at a 3D scene.



[5]

The Helper package is a small package providing various primitives and abstractions that make things a lot easier in other packages.

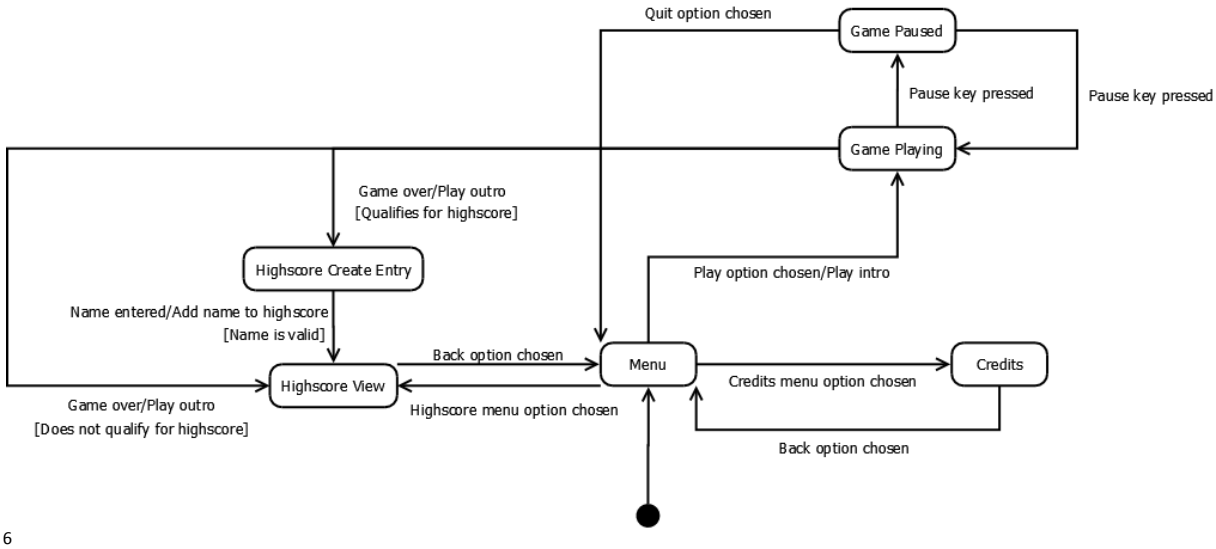
[5] HelperClassDiagram.png

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

View

Classes that are concerned with outputting data to the user.

- PacmanGame
  Main game class, runs the actual game loop.
- GameEventSubscriber
  An interface used in the model package to notify the view of certain game-specific events.
- Screen
  The base class for the different game screens.
  - MenuScreen
    The main menu, where the game starts.
  - HighscoreScreen
    The highscore screen, meant to present the highest scores of past players. Can also insert a new entry into the list of scores.
  - CreditsScreen
    A screen mentioning the development team.
  - IngameScreen
    The screen where most of the game happens. Draws the scene and updates the model.



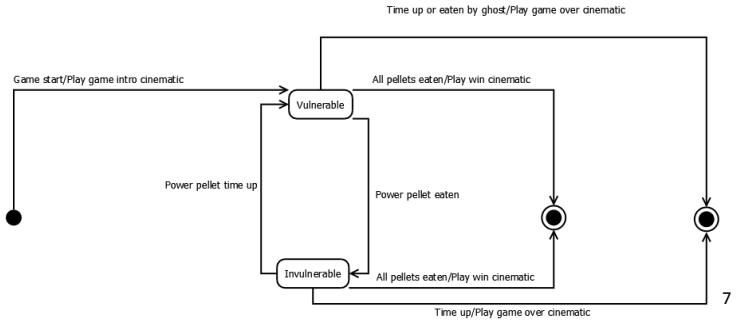
6

- GUI
  Components that make up the graphical user interface in the screens.
  - Menu
    A list of selectable options.
  - Table
    A table of data.
- Scene
  The component that draws the actual game. Collects data from the model and presents it graphically.

[6] UIStateMachine.png

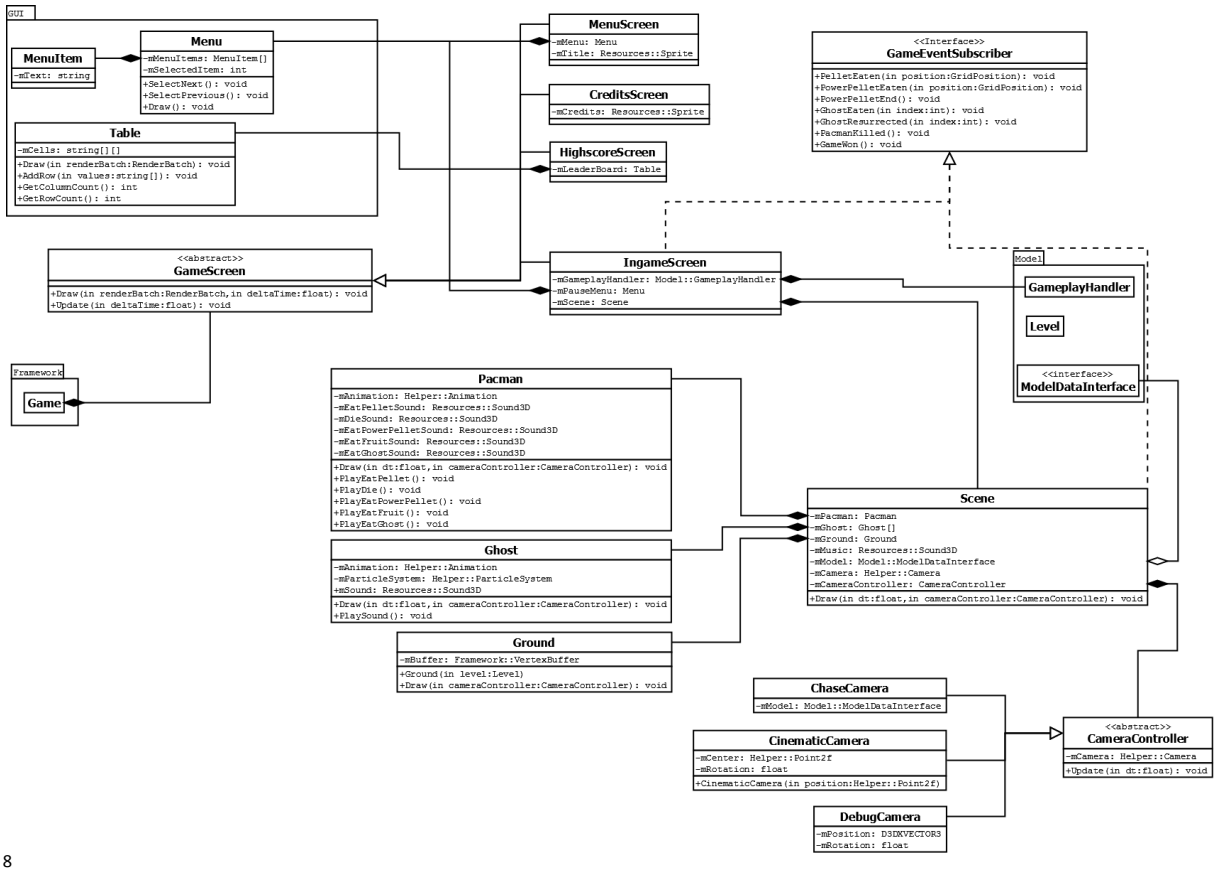
thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

- Ground
  Draws the tiles of the level, walls and floors.
- Ghost
  Draws a ghost, with an animation/model and a particle system.
- Pacman
  Draws Pacman, with an animation.



[7]

- ChaseCamera
  A camera controller, chases Pacman's position, and can be turned to look either forwards or backwards.



[8]

[7] PacmanStateMachine.png
[8] ViewClassDiagram.png

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

The View package is the largest package in the system, responsible for all output to the user. This includes 2D rendering such as GUI and HUD elements, 3D rendering such as the actual Pacman game, and sound output, both in two and three dimensions.

The View package communicates with the Model package using the Observer pattern through the GameEventSubscriber interface, and also collects the remaining data necessary from an interface in Model called ModelDataInterface.

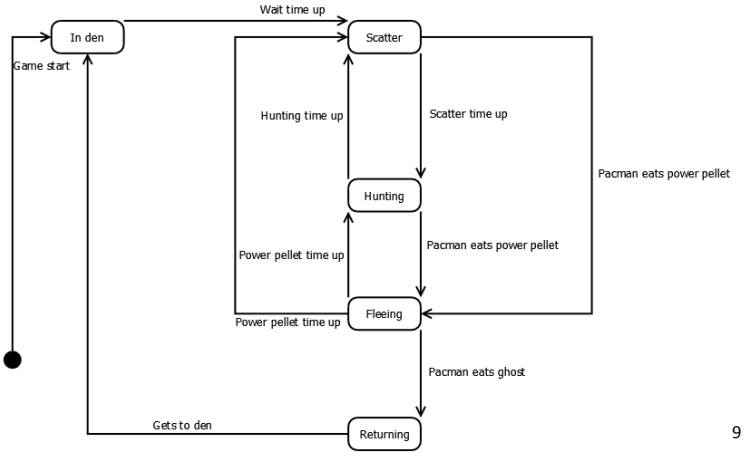
To control the camera the Strategy pattern is used, since there are three very specific behaviors a camera can have.

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com

Model

Contains the classes concerning the logic of a Pacman game session.

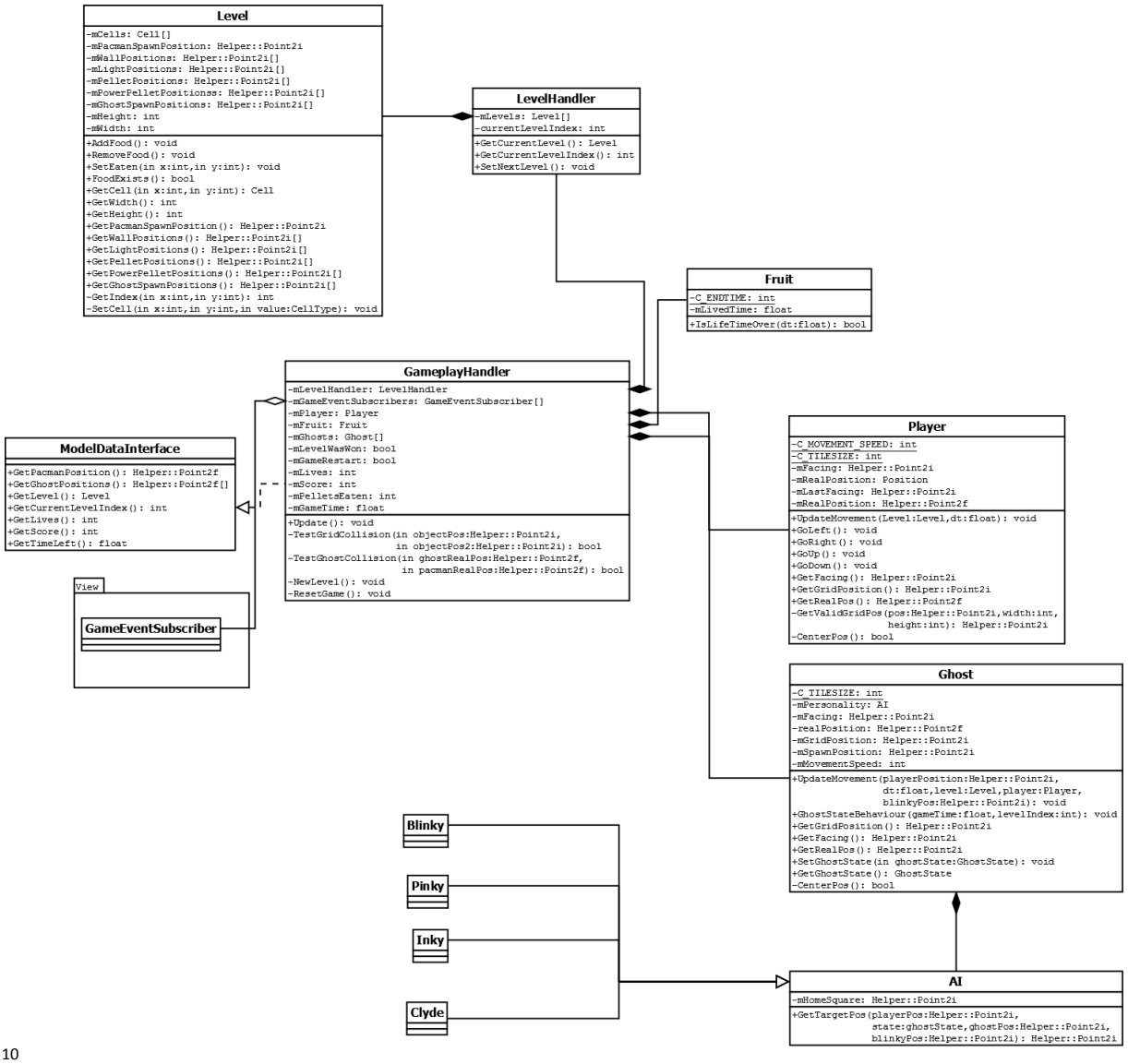
- GameplayHandler
  Handles one session, moving through all levels until game over.
- ModelDataInterface
  Exposes data to the view, necessary to render the game.
- LevelHandler
  Handles the progression through the levels.
- Level
  Contains the data of one level. Is able to load itself from a .png file.
- Player
  Handles the movement and logic of Pacman.
- Fruit
  Handles the fruit bonuses that sometimes appear in the game, with a timer.
- Ghost
  Handles the logic of the enemies in the game. Contains an AI object, which can be used to change the behavior with the strategy pattern.
- AI
  Determines the behavior of a ghost. The following AIs will be implemented, one for each ghost:
  - Blinky
    Moves towards Pacman's actual position.
  - Pinky
    Attempts to move in front of Pacman.
  - Inky
    Tries to move to its mirror position in regards to Pacman's path.
  - Clyde
    Moves towards Pacman, but flees when he gets too close.



[9]

[9] EnemyStateMachine.png

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com



10

The Model package is where the game is run. Due to the crisp distinction between the View and Model packages, this package could essentially be reused with a completely different view without any modifications. Naturally, the reverse should also hold true. The Gameplay Handler component calls certain methods in its GameEventSubscribers when certain events occur that require the View to display particular effects, otherwise it supplies the View related information by implementing the Model Data Interface.

For the ghost AIs, Strategy pattern has been chosen. The difference between the ghosts is how they decide their target position. That one method is what separates them, so a Strategy pattern is tailor made for this.

[10] ModelClassDiagram.png

thsi08@student.bth.se, martin_svart@hotmail.com, lars.woxberg@gmail.com,
kim.restad@gmail.com, fredrik.johannesson@hotmail.com