**N.Y Tajms Studios proudly presents!**

**a tajms opengl production**

**pacman 3d**

**Group Members:**Eric Ahlström  
John Bergsten  
Lucas Holmqvist  
Christian Markowicz  
Konrad Öhman

Contents

[Introduction 2](#_Toc378603815)

[Gameplay introduction 2](#_Toc378603816)

[Architecture Diagram 3](#_Toc378603817)

[Game-specific Subsystem 4](#_Toc378603818)

[Front-end System 4](#_Toc378603819)

[Gameplay Foundations 5](#_Toc378603820)

[Visual Effects 5](#_Toc378603821)

[Low-Level Renderer 5](#_Toc378603822)

[Debugging 6](#_Toc378603823)

[Collision and physics 6](#_Toc378603824)

[Audio 6](#_Toc378603825)

[Human Interface Device 6](#_Toc378603826)

[Resources 7](#_Toc378603827)

[Core Systems 7](#_Toc378603828)

[3rd Party SDKs 7](#_Toc378603829)

[OS and Platform 7](#_Toc378603830)

[Domain Modell 8](#_Toc378603831)

[Work Breakdown Structure 9](#_Toc378603832)

[References 11](#_Toc378603833)

# Introduction

Pacman is one of those ironical video games produced during the industry’s infancy. No one calls him/herself a gamer without having at least heard of the legend that is Pacman. We are thrilled to now be tasked with the creation of our own version. This document is the basic conceptual outlines of what we hope our game will be.

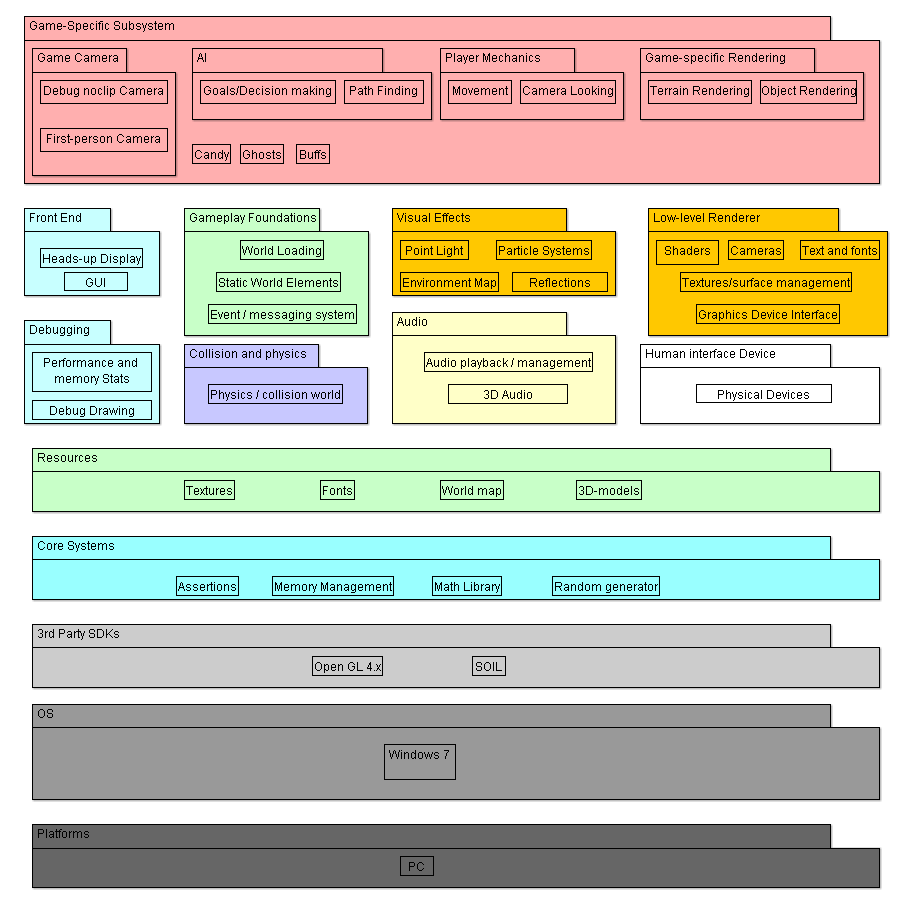
# Gameplay introduction

The player plays as Pacman from a first-person perspective, albeit in a 3D environment. The goal, as with the classic Pacman-games, is to eat all the candy while avoid being eaten by the ghosts. One fundamental change is the layout of the map. The player has to navigate on paths surrounded by black pits which Pacman can fall down into and die.

Aside from the dangers of falling down from the paths, Pacman will starve if he fails to continually eat candy. This has consequences: Pacman’s movement speed will be reduced and the light will begin to fade, which will make spotting and running away from ghosts significantly harder.

# 

# Architecture Diagram

This model is used to show the various system components which will constitute our game. These are coupled to different classes detailed in a domain model later in the document.

## Game-specific Subsystem

This block contains components that are specific for this game, such as the candy and ghosts.

**Camera** contains two cameras: a noclip-camera to easily fly around the world for debug purposes and a first-person game-camera for the final gameplay.  
**AI** directs the ghosts’ movement by sending a path to each ghost. It also tells the ghost whether to flee from Pacman or to chase him.   
**Player mechanics** manages play-inputs from the keyboard and mouse which are then used to control Pacman and for navigation of menus.  
**Game-specific Rendering** renders the terrain and different objects.  
**Candy** aresmall pieces of candy which Pacman eats.  
**Ghosts** are the main enemy. They follow a path given by the AI, based on their state and Pacman’s position.   
**Buffs** changes the state of Pacman. The most basic buff is that he can eat the ghosts for a brief period of time.

## Front-end System

Contains HUD and GUI. These are things that don’t interact with the gameplay but rather shows info that is useful to the player and directs the player to different places of the program.

**HUD** (Heads Up Display) contains essential information such as lives and score which is then presented on the screen during gameplay.  
**GUI** (Graphical User Interface) includes different menus which navigates the player to different places in the program.

## Gameplay Foundations

This block contains map loading, different static objects (like spawn points) and an event/messaging system.

**Map Loading** reads the map and saves both a logical representation of the map for collision detection and path finding, and a graphical representation that is sent to the graphic interface.  
**Static World Elements** are elements/objects that are in the same place throughout the game, like spawn points.  
**Event/Messaging System** gives the player information about gameplay states, for example when the ghosts changes state from fleeing to chasing.

## Visual Effects

Different lighting and special effects.

**Point Light:** a light source emitting light from one point. It is the main source of light in our game.  
**Particle Effects** is a way to demonstrate small particles with different movement patterns. Explosions are represented by particle effects.  
**Reflections** put a texture on a surface that represents the view of the surface.  
**Environment Map** controls background textures.

## Low-Level Renderer

Contains lower levels of rendering, such as shaders.

**Shaders** are programs that reside in the GPU. They tell the GPU how to color each pixel seen by the camera.  
**Cameras** are matrices that convert the world into different views.  
**Text And Fonts** are outputs to the GUI and can also be used for debugging. The fonts tell the text how to display itself.  
**Textures/Surface Management** puts the correct texture on the right object.  
**Graphics Device Interface** is the interface towards the low-level renderer.

## Debugging

Is used to check the program code for bugs, glitches and performance issues.

**Performance and memory stats** keep stats about performance, such as frames per second.  
**Debug Drawing** prints the map and objects in different ways. Can be used for drawing only the outlines of polygons.

## Collision and physics

Used to check different forms of collisions.

**Physics / Collision World** use positions to check if collisions occur.

## Audio

Plays sounds and music depending on the position of Pacman.

**Audio Playback / Management** controls what sounds to play and what volume it should be played at. The volume depends on the distance from the audio source and Pacman.  
**3D Audio** makes the sound appear different in each output device.

## Human Interface Device

Mouse and keyboard are the only interface devices that the player can use.

**Physical Devices.** The keyboard is used to control Pacman left, right, forward and backward. The mouse is used to control the camera and Pacman’s direction.

## Resources

Useful files for our program.

**Textures:** files with colors that we can attach to an object.  
**Fonts:** information that describes the way a text should be written.  
**World Map** stores information about how the vertices are to be constructed and how the logical map will be contracted. It is read from a .raw map.  
**3D-models** are stored in .obj files with information on how to construct the vertices for a 3D-object.

## Core Systems

These are the different libraries which help us by providing predefined methods.

**Assertions** check the code for logical errors like misspelling or forgotten characters.  
**Memory Management** helps with deleting memory places that won’t be used again. This makes memory loss less frequent.  
**Math Library** provides structures and mathematical calculations.   
**Random Generator** uses the CPU clock to give the program a seemingly random number.

## 3rd Party SDKs

Software programmed by a 3rd party which we make use of in our program.

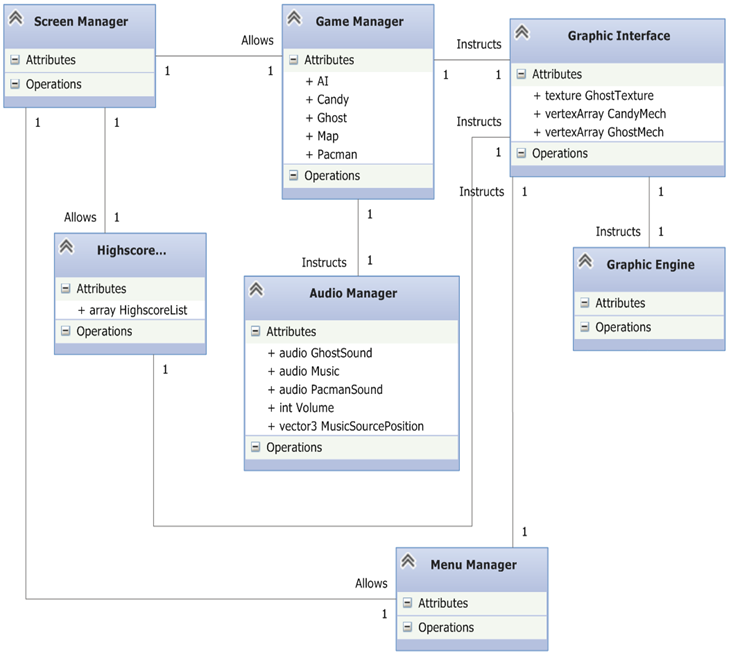
**Open GL 4.x** is the SDK that will be used to manage the GPU.

**SOIL** reads textures.

## OS and Platform

This game will be programmed for Windows 7 using a pc.

# Domain Modell

This modell is used to more specifically show an overview of the classes which we will implement, in other words classes to be included in the game. Various classes are contained within specific components, detailed below. Keep in mind that the Domains will be composed of many other classes (for instance, the Game Manager will have an entire class dedicated to AI).  
  
//domain model is changed. Camera removed as it is part of the graphics engine component

## Screen Manager

Manages inputs from the ***Human Interface Devices***, and sends them further down the chain to the different managers.

## Game Manager

Manages the overall game. Uses all ***game-specific subsystems*** except those related to the cameras and the rendering. The game manager also makes use of **the gameplay foundations** to aid the subsystems. **Debugging** information is also collected.

## High Score Manager and Menu Manager

Uses none of the architecture components explicitely, although collects information and sends to other components (such as those associated with ***rendering*** or ***audio***).

## Audio Manager

Contains ***audio***.

## Graphic Interface

Works as an interface to the engine. Contains ***game-specific rendering*** and ***game*** ***cameras***. Instructs the Graphic Engine on how to use its shaders, textures etc.

## Graphic Engine

Recieves instructions from the Graphic Interface on how to utilize ***low-level rendering*** and ***visual effects***. Makes use of ***resources*** and ***3rd party SDKs***. Also displays ***debugging*** information.

# Design Patterns

We have decided to use three design patterns from the course litterature: Singleton, Façade and Observer.

## Singleton

We are using a singleton pattern for our graphichandler class. We do not want more than one instance of our graphichandler because we want the draw calls from the screenmanager to be as simple as possible. The screenmanager initializes the other managers (highscore, game and menu) and gives them a pointer to the graphichandler. This will allow the managers to call for their needed drawmethods easily by using the pointer to the graphichandler.

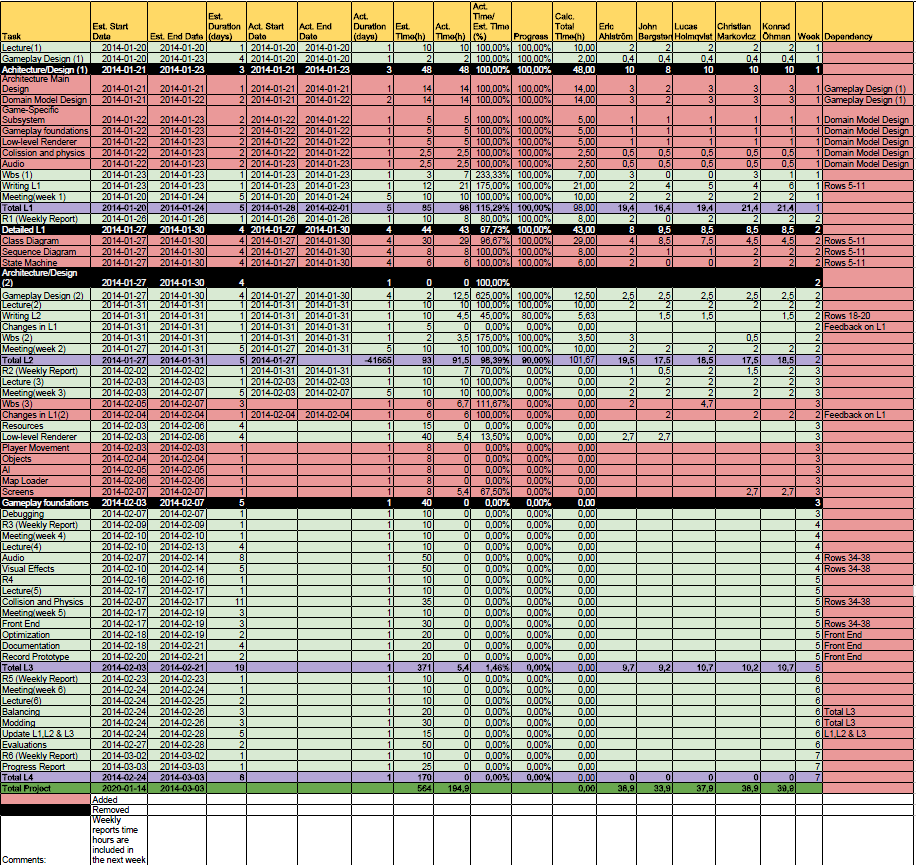
## Façade

The graphichandler class also implements a façade pattern as it manages all of the graphics classes below it without the user needing to know how the subsystem works. However the handler does not hide the subsystem completely so that the programmer has the ability to get a broader knowledge of how the program works.[[1]](#footnote-1) We hope to use this code for future projects but still want to be able to change the code if needed.  
 We got this idea from C# XNA programming where the programmer only has to call the spritebatch class to be able to draw objects to the screen. We thought this was a very simple and easy technique to get graphic.

## Observer

The gamemanager class has an instance of every objectclass below it (candies, ghosts and pacman). All these classes affect eachother and therefore we need a class that can notify them of changes in the gamestate i.e pacman eats a supercandy and is allowed to eat ghosts.  
 The use of an observer gives us a loose coupling between our objects while still giving us the ability to notify all of them without the objects knowing of eachother.[[2]](#footnote-2)  
 By coupling all of our objects to the gamemanager we reduce the amount of coupling a lot however it puts a lot of work onto one class. The gamemanager has other tasks than notifying the objects of changes in the gamestate such as collision detection. Therefore we are unsure of the gamemanager being called an observer.

# Work Breakdown Structure



# Class Diagram

The class diagrams are an elaboration of the domain diagram.

## Gamelogic Class Diagram

# E:\3dpacman projekt\Pacman\GameLogicClassDiagram.png

# 

## Key Game Logic Concepts

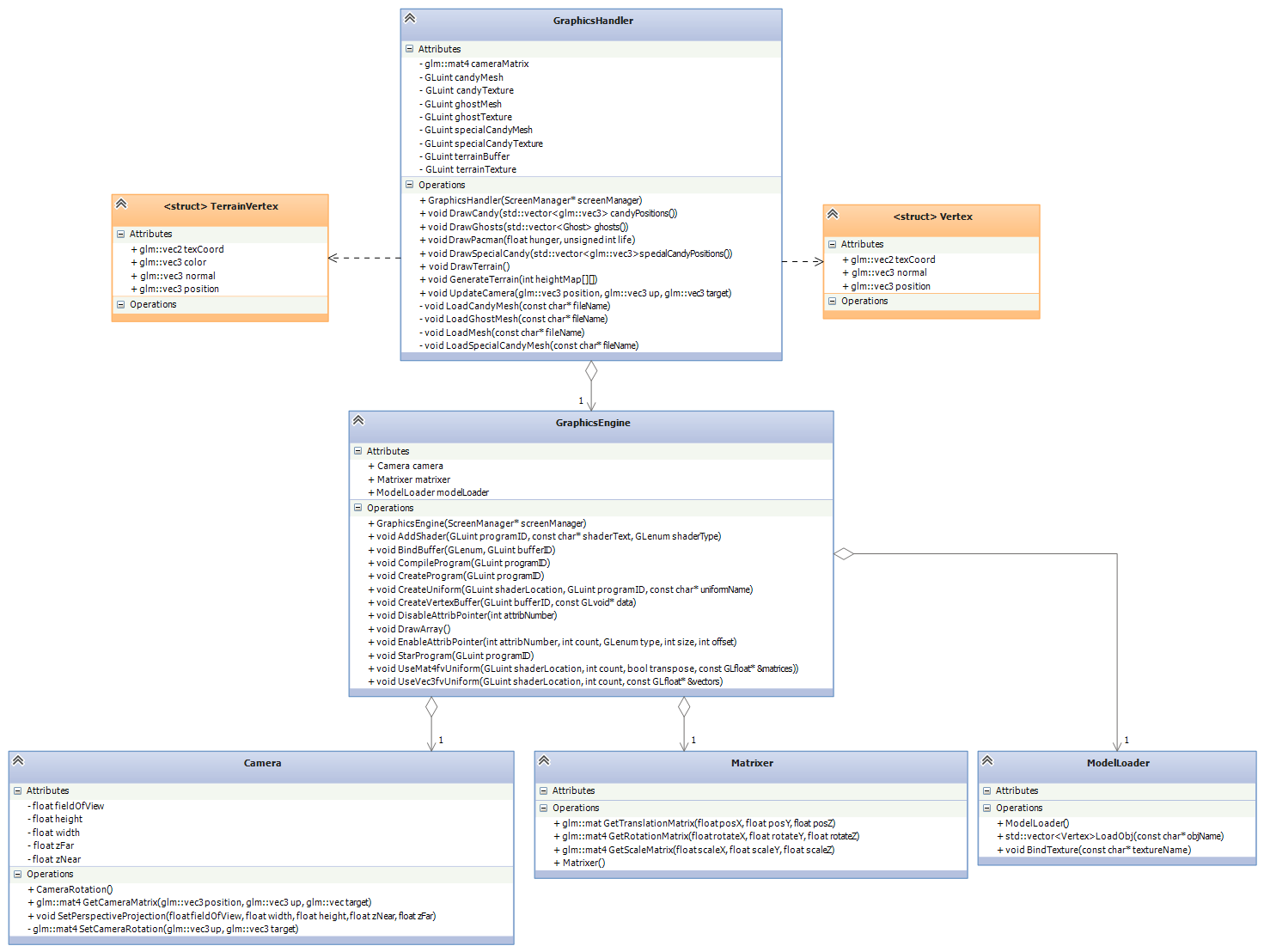
This is an explanatory text detailing the key parts of our game logic engine.

## Maps

Maps are built using simple height map techniques. A .raw file is loaded from which a two-dimensional array is built with the values from the .raw files. From this map, the values are used to generate the different objects in the world. For instance, one specific value means a spawn point for a ghost and another means the location of a power up.   
 The map is also used to generate the 3D vertices. Any value above zero should be interpreted as a vertex composing the world on which Pacman can move. Zero-values compose the pits which Pacman risks falling into.  
 Collision detection also makes use of the map.  
  
AI  
The AI uses Dijkstra's algorithm[[3]](#footnote-3) for path finding. The ghosts will get an updated path every 5-10 seconds. This makes the ghost aim for a spot that Pacman has already left making the path seem a bit random.  
When the ghost gets to his goal (the position where Pacman was) he will immediately get a new path. This makes it possible for the ghosts to catch Pacman.  
 We will also add a random generator making it possible for the ghost to head to a position where Pacman isn’t. This makes it harder for the ghosts to group up and making it available to play the game without dying really fast.

Collision Detection  
There are two different types of collisions: when Pacman walks into a pit and one where Pacman collides with an object (ghosts or candy).  
 The collision between Pacman and a pit will be determined by checking whether or not Pacmans position in the Y-axis (vertical) has a valid value. If the value for example is below a certain point pacman has fallen down into a pit.  
 The collision between Pacman and an object is determined by checking Pacmans position and the objects position. If they are in close enough proximity, it’s a collision

## Graphic Class Diagram



Graphics  
The graphics engine is designed to be as independent as possible, to be able to be used for other projects and to be easily modified/upgraded. In order to use the engine, a graphics handler is used. The graphics handler is more specific for the game, containing game-specific shaders and vertex buffer calculations.   
 To summarize: the handler calculates the vertex buffer and the uniform valuables then sends it to the engine, which simply binds any buffer or uniform it receives. The camera works similarly. The Graphics Engine and its subclases constitute the low-level renderer and the graphics handler with the structs is the game-specific rendering component.

# Class Description

The following is a short description of each class we are going to use in our project.

## Screen Manager

The screen manager is the highest heirarchial class. It manages three states (screens): The main game, highscore and the menu. It will receive data from usercmds and send it down to the currently active screen.

## Menu Manager

Menu will consist of a couple of clickable buttons which will return a trigger for the screen manager to change state when clicked.

## Game Manager

This is our main game logic class. It will create an instance of each object in the project, call their updates, send neccesary information for statechanges and call the graphichandler to draw.  
 It will also manage the collision detection, load the map and make the logical map for candy placements.

## Audio

We will use openAl to create sounds. We have no previous experience with this SDK so we’re not sure how it works yet.   
 The class will receive the position of pacman from game manager to calculate the volume from each soundsource.

## Pacman

Pacman is the player and our game is in first person. Therefore pacman will contain camerainformation.  
The class will update its position accordingly to the information from usercmds and the updated position will be sent back to game manager for collisiondetection.

## Ghost

The ghostclass will update its position and direction accordingly to the pathfinding algorithm and a state which will be updated after the information send down from game manager.

## Matrixer

This class will manage all the matrices needed for camera and world transformations.

## Model Loader

The model loader will give the graphicsengine a tool to load meshes. This is where our SOIL sdk will be utilized.

# Deviations from project plan

This week we slipped behind two and a half days. In our initial plan we made an assumpion that our customer would give us feedback early. We now understand that this is not how it works, next time we will ask the customer first when we want to schedule customerfeedbackdependant tasks ahead of time. We did not plan to adjust our first draft this much which also put us behind. The customer has mentioned he will give us feedback on L2 as soon as possible and our codework has started.   
Hopefully we don’t have to change much after the L2 feedback or we have shot ourselves in the foot again.  
We reasoned at the group meeting that we would start coding anyway since building the skeleton of the code is better than just waiting for the feedback to arrive.

# References

We have used the course main literature as a template for our architecture diagram. We have removed some of the content that they had since we found it unnecessary for this project. [[4]](#footnote-4) For example we decided to not include skeletal animations on the basis that it is pointless in such a simple game as Pacman.

1. Design Patterns: Elements of Reusable Object-Oriented Software E.Gamma, R.Helm, R.Johnson, J.Vlissides, 1994 Addison-Wesley, pg.208 [↑](#footnote-ref-1)
2. Design Patterns: Elements of Reusable Object-Oriented Software E.Gamma, R.Helm, R.Johnson, J.Vlissides, 1994 Addison-Wesley, pg.327 [↑](#footnote-ref-2)
3. http://www.csc.kth.se/utbildning/kth/kurser/DD2458/popup11/dokument/graphs2.F6.09.pdf [↑](#footnote-ref-3)
4. Game Engine Architecture, Jason Gregory, 2009 Taylor and Francis group, p.29 [↑](#footnote-ref-4)