JoHanna Kirchmaier
Anja Bergmann
Tabea Halmschlager

HATman

Project Work 1

ITM14

Content

Introduction

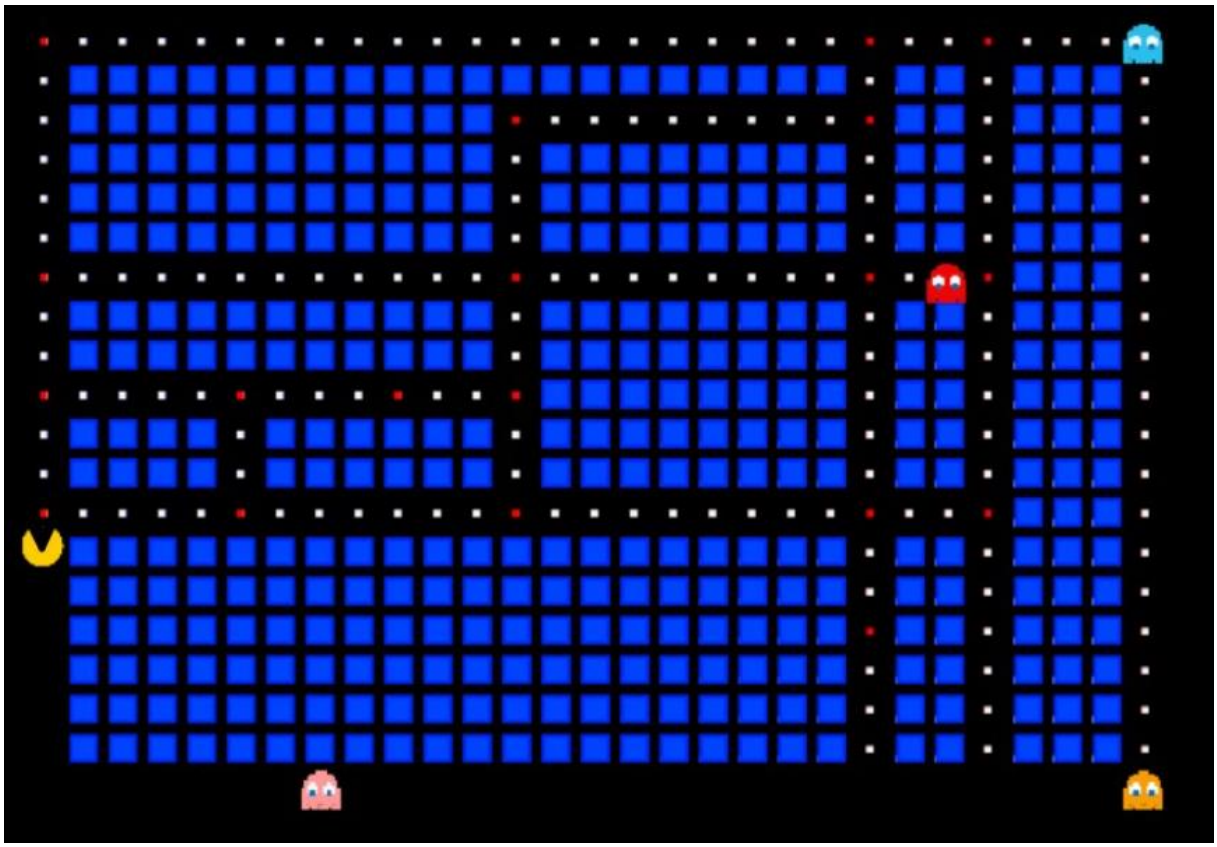
https://upload.wikimedia.org/wikipedia/commons/thumb/0/0a/Python.svg/2000px-Python.svg.png

Python has many facets. The term does not only refer to huge, nonvenomous snakes and a british comedy group, but also to a high-level programming language. Though it is usually rather known as a scripting language, we wanted to find out if it also is suitable for the purpose of writing a multiplayer game. To anticipate the result: It turned out to be much more complex than we thought, of which more later. Something to look forward to, since we will be happy to describe every single terror that occurred to us during the project.

The main goal of our project was to implement a game that was similar to the iconic game Pac-Man. Instead of the classic single player version, we wanted to make it possible to play the ghosts as well. For this matter we needed not only a graphical user interface, but also a way of communication between the players. We decided to run the game clientsided and send updates of the course of the game to a server which would inform the other clients about the changes.

Gameplay

(For those who do not know how Pac-Man works.)



Urbandictionary description for Pac-Man: A yellow pizza missing one slice. He apparently has no eyes, lives in the 2nd dimension, has a wife that wears lipstick and a bow, and has a child that wears a beanie. He eats pellets and runs from ghosts. On lucky days, he eats pellets that enable him to devour those same ghosts. Occasionally, he eats fruits...or pretzels. xD

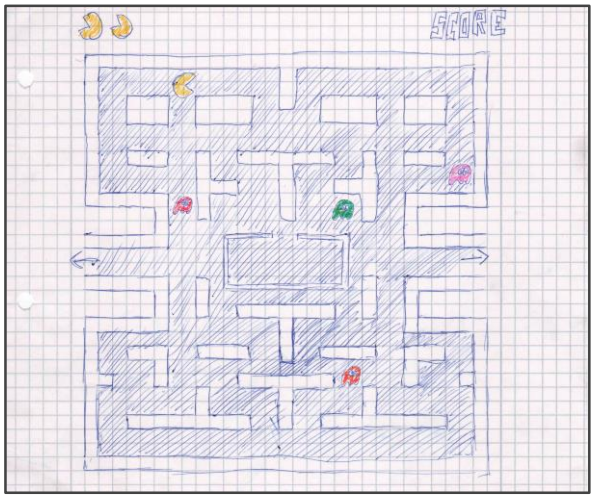
Normally the player controls Pac-Man through a maze to eat yellow dots. When all dots are eaten Pac-Man gets to the next stage. Four ghost enemies (Blinky, Pinky, Inky and Clyde) try to catch him, if an enemy touches Pac-Man, he loses a life. If he has no lives left, the game ends.

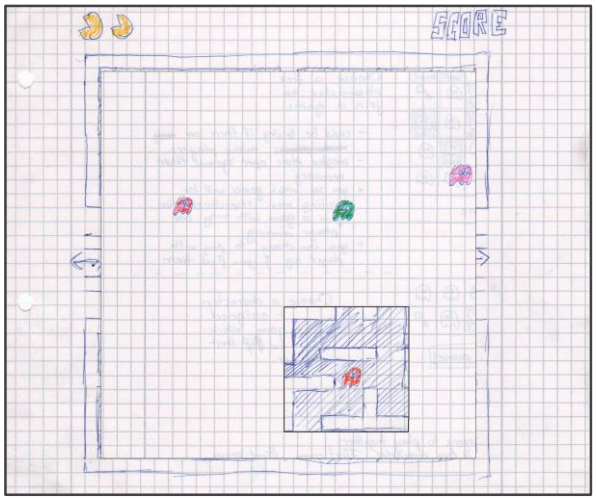
There are larger dots called power pellets that give Pac-Man the temporary ability to eat the ghosts. When eaten, the ghosts regenerate in their normal colour in the center box of the maze. Shortly before becoming dangerous again the ghosts flash white.

However in our version you can also play as the ghosts, making it your goal to catch Pac-Man.

Approach

Before we started to torment our keyboards, we racked our brains about some basic questions of the high philosophy of pacman gameplay and programming, skimmed through a bachelor thesis concerning the framework cocos2d and whipped out our pencils to design some artistic paper prototypes which might get auctioned for a worthy cause sometime when we are rich and famous and every child will know the HATman corp.





But short time later, it became complicated. Our project turned out to be much more complex than we thought in the beginning. Especially the issue to transform a single player game into a multi player game proved to be quite tricky. On the way from our first google search concerning the topic to the final project we encountered a lot of problems. Additional to the below described attempts, we also considered and/or tried numerous other frameworks, but discarded them since they were either deprecated, only available for a older python version or poorly (respectively not at all) documented. Our main approaches are described below - more detailed descriptions of the frameworks can be found in the chapter “Frameworks”- , while the others are omitted due to the fact that describing all of them would go beyond the scope of this documentation. Just be assured, it was a long and treacherous journey all along.

Attempt#1 Cocos2d

Our first attempt was to use the framework Cocos2d. We managed to make the game playable, but had problems when it came to implementing the multiplayer mode. We found a lot of frameworks that provided the functionality we needed, but unfortunately they were all written in Python 2 and therefore not 100% compatible with Python 3, which we chose because we wanted to work with the latest versions.

This is why we condemned everything we had so far and started all over. This time we intended to use the framework Arianne.

Attempt#2 Arianne

With the help of Mr. Knoll, we found a tutorial for the Arianne engine Marauroa (*https://stendhalgame.org/wiki/InitialStepsWithMarauroa*) which seemed to explain everything we wanted and by means of which we tried to implement a mulitplayer version of our first attempt.

The following pages describe the numerous steps we took to try and make Marauroa work, the errors we had to solve and problems we had to work around. In the end it all failed miserably and resulted in us going back to attempt #1 with the addition of the framework Twisted, but more about that later.

So here's the story ...

We downloaded Marauaroa from https://github.com/arianne/marauroa and built it with Ant:

- ➢ ant dist

Then we imported it into the Eclipse workspace, to have some help finding syntactical errors.

We set up the database:

- ➢ mysql –user root –password
- ➢ Password: toor
- ➢ Create database marauroa
- ➢ Grant all on marauroa.* to hatman@localhost identified by ‚keines'
- ➢ Exit

The server needed a configuration file called server.ini which we had to create (see https://stendhalgame.org/wiki/Configuration_file_server.ini) and a file named marauroa.ini (see https://stendhalgame.org/wiki/How_to_write_games_using_Arianne_in_Python), which we tried to create to the best of our knowledge.

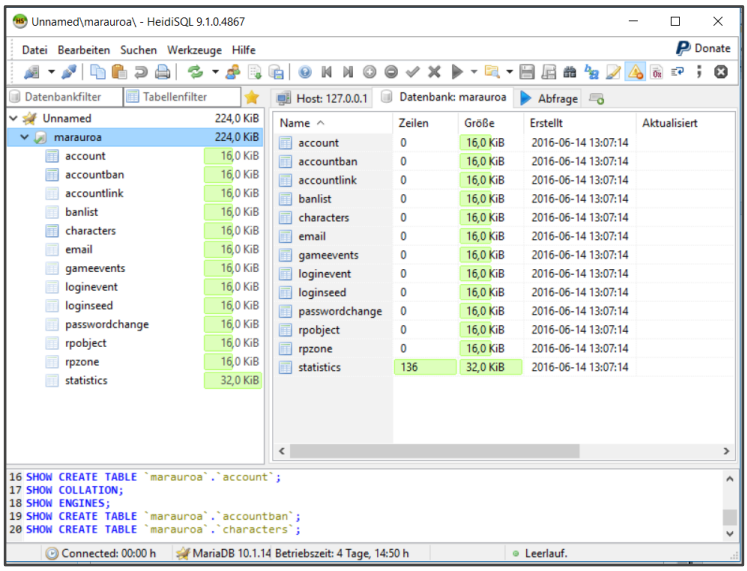
Then we downloaded the MariaDB-Connector (https://downloads.mariadb.org/connector-java/) and include the jar in our project, by putting it into the "libs"-folder and adding it to the Java build path.

Then we could start the marauroa-server (src > marauroa.server.mararoad.java).

```
Starting Marauroa https://arianne-project.org/engine/marauroa.html
Arianne's open source multiplayer online framework for game development
Marauroa is released under the GNU General Public License: LICENSE.txt

Configuring Log4J using marauroa/server/log4j.properties
INFO  [main      ] AbstractDatabaseAdapter  (122 ) - Connected to jdbc:mariadb://localhost/marauroa: MySQL 10.1.14-MariaDB
INFO  [main      ] AbstractDatabaseAdapter  (122 ) - Connected to jdbc:mariadb://localhost/marauroa: MySQL 10.1.14-MariaDB
INFO  [main      ] AbstractDatabaseAdapter  (122 ) - Connected to jdbc:mariadb://localhost/marauroa: MySQL 10.1.14-MariaDB
INFO  [main      ] AbstractDatabaseAdapter  (122 ) - Connected to jdbc:mariadb://localhost/marauroa: MySQL 10.1.14-MariaDB
INFO  [main      ] UpdateScript             (122 ) - Checking database structure and updating it if needed.
INFO  [main      ] UpdateScript             (122 ) - Completed database update.
INFO  [main      ] marauroad                (122 ) - marauroa 3.9.5 is up and running... (startup time: 0.3 s)
INFO  [marauroad ] Statistics               (122 ) - Total/Used memory: 251392/24907
```

The database was automatically configured. Therefore we could see that some tables were already created in the database by Marauroa:



As we didn't know how to proceed from there, we went through different readme files included in the Marauroa directory and through numerous pages of the documentation wiki. Some pages seemed deprecated, others were very confusing or just led to various similar pages so we had no idea which steps we needed to perform next.

To gain some clarity about how Maraurora is supposed to work and to have a model which we could hopefully orientate towards, we tried to install Mapacman - the multiplayer Pacman game described in the tutorial mentioned before.

We downloaded it as a tar file from https://arianne-project.org/game/mapacman.html and unpacked it. Starting the programme with Python 3 didn't work, so we tried to upgrade the files with the Python script 2to3.py.

- python34 C:\Python34\Tools\Scripts\2to3.py –w mapacman.py
- python34 C:\Python34\Tools\Scripts\2to3.py –w sprites.py
- python34 C:\Python34\Tools\Scripts\2to3.py –w theme.py

With the next attempt to start mapacman.py with Python3 we got an Import Error:

- „ImportError: No module named ‚pygame'"

Since installing pygame via the command "pip3 install pygame" would not work, we downloaded it from http://www.pygame.org/download.shtml. We got another Import Error. This time it was the module names.pygame.base that could not be found. Being clueless where to get it from as google could not help us, we gave up on Mapacman with Pygame and downloaded a chat with Pyarianne instead.

The chat client did not work because of some wrong indentations that we fixed by manually going through the file, adding and deleting indentations where needed or needless.

After also mending the print statements that did not work, we got another error. The imported module pyarianne was trying to import the module arianne which was still missing. We were not able to find the module and finally gave up on Arianne and Marauroa.

Attempt#3 Twisted

After we wasted a huge amount of time and effort on the hopeless task of trying to set up a programme with Marauroa and its sub-modules, we eventually resumed our attempt with cocos2d. This time we planned to include a framework called Twisted to handle the networking part of the game.

We found a detailed explanation of the framework and its mechanics (http://krondo.com/an-introduction-to-asynchronous-programming-and-twisted/) and used the sample code as foundation for our own game server.

The GUI is made up of cocos2d layers and sprites, which the player can move with the arrow keys. Every time the player moves his character, the client sends a message to the server which then notifies all other clients of the change so they can update their GUI accordingly.

Frameworks

From the fact that we are programmers and programmers are lazy, follows that we are lazy too. But in the case of programmers, laziness is not a weakness, but a virtue. Due to this laziness we just logically proved, we planned not to implement everything on our own, but use some frameworks instead.

Right from the start we decided to use Python 3.x as well as the framework Cocos2d for our graphical user interface. We came across Cocos2d because our tutor had a student with a bachelor thesis about this framework combined with game development. Nonetheless, we still needed a way to connect the graphical user interface with a client-server-architecture for communication between all players. After a lot of different approaches (especially with the framework Arianne), we eventually decided to use Twisted, which will be described in the following.

In addition to our choice of programming language and frameworks, we also had to update our existing Windows versions to Windows 10, since the Cocos2d sprite object did not work properly on Windows 7.

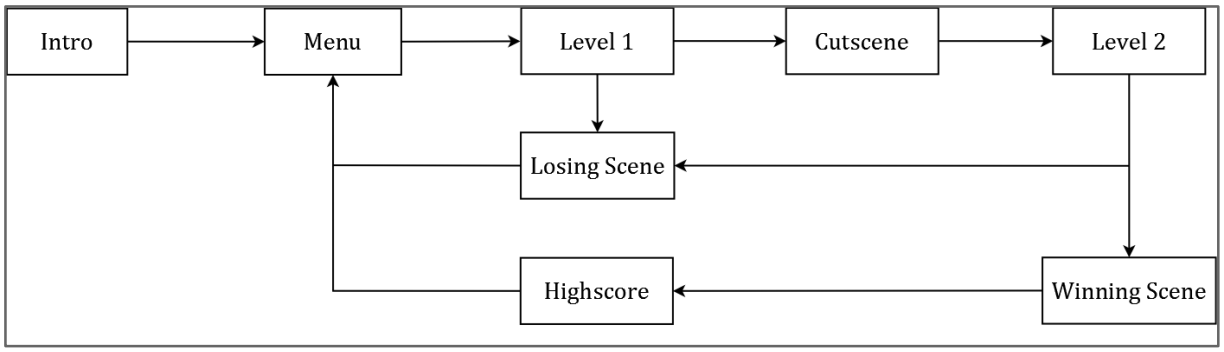
Cocos2d

Cocos2d provides some basic design components to work with, such as sprites, layers and scenes and is therefore perfect for building 2D games, demos, and other graphical and interactive applications. The framework builds up on the Python library pyglet and thus, importing that is also necessary.

For basic understanding it is crucial to know that cocos2d has some fundamental concepts (CocosNode, scenes, director, layers, sprites and event handlers) implemented and these are used by every cocos2d application. The most fundamental concept would probably be the **CocosNode** element. It is basically the main class of all other elements including components like a scene, a layer or a sprite.

A **scene** is a more or less independent piece of the app workflow and is also called a “screen” or “stage”. An application can have many scenes, but only one of them is active at a given time. For instance, most games would have following scenes:

- an intro with a title, the version or copyright text,
- some kind of menu in which you can choose a character, a name, start/save/exit the game, etc.,
- your last or the first level of the game,
- a winning and losing cutscene
- and maybe also a high score table scene.



As you can see in this example, you can define every scene more or less as a separate app, although there is a bit of glue between them containing the logic for connecting scenes, which is illustrated by the arrows. The Intro here goes to the Menu when interrupted or after finishing, and if you start a game, the Menu scene would be deactivated and the Level 1 scene activated.

In order to switch between different scenes, a **director** is needed. The director is a shared singleton object and the component which takes care of going back and forth between scenes. It always knows which scene is currently active, because it handles a stack of scenes to allow for instance a "scene call". By performing a scene call, it is possible to pause a scene, to put it on hold while others are entering and afterwards to return to the original scene. Thus, the push, replacement as well as the end of a current scene are all managed by the director. Moreover, this component is responsible for initializing the main window via *director.init()*.

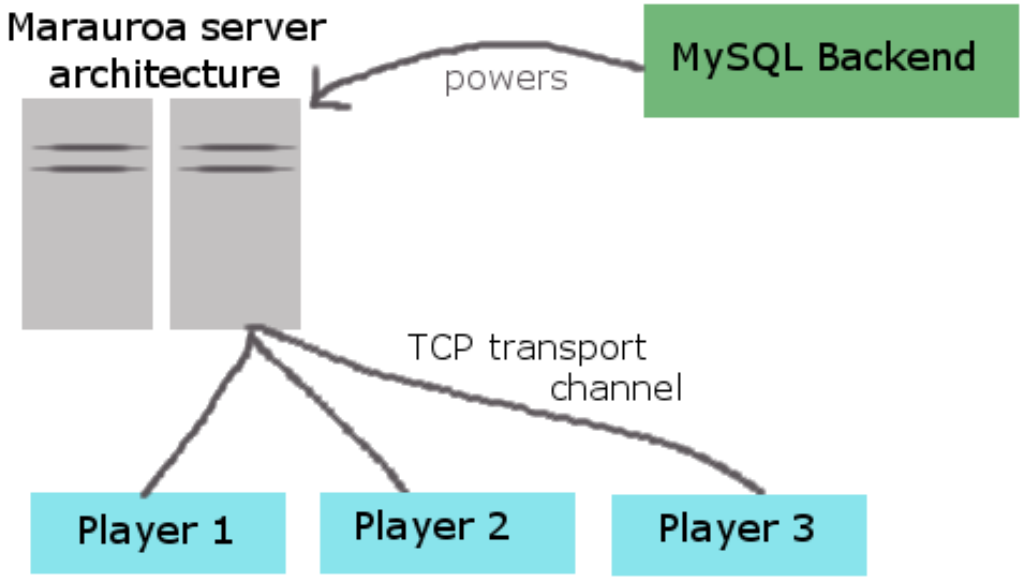
What is also crucial for working with cocos2d, is a so-called **layer**. Layers are the most near descendants of a scene node because they hold and organize individual elements for a scene. One can think of layers as usually transparent sheets where the children are drawn. The scene per se would then be like a stack of sheets put together from the back to the front axis. As you can therefore imagine, layers are in charge of defining appearance and behaviour. Take, for instance, event handlers: Events are propagated to layers from front to back until some layer catches the event and accepts it.
Another key thing to remember about layers are their subclasses which already provide a good basis for further developing. One of them is the Menu subclass and it is certainly useful for implementing simple menus that are indispensable for a game.

But what would be a good game without characters that wander around on your screen whenever you want them to? - Exactly, that would not be a good game. So it is obvious that you need not only event handlers but also some kind of **sprites**.
A cocos2d sprite is actually like any other computer sprite because it is a 2D image that can be moved, rotated, scaled, animated, transformed and so on. Sprites are implemented using the Sprite class (a subclass of CocosNode) and can have other sprites as children. Thus, these children are transformed, when their parent sprite is transformed.

As already stated above, cocos2d also allows you to use **event handlers** on the basis of the pyglet event framework to react to events of the programme flow and to perform actions. One instance of the pyglet.event.EventDispatcher is the emitter. Each emitter registers as much events as desired and each one is identified by a string called the event name. But in order to really react to events, listeners must be registered with the emitter. Essentially, the emitter is provided with a (<event name string>, callable_func) and will call the callable_func when <event

name string> happens. Important to know is that any number of listeners can register for a (emitter, <event name string>)-pair and that a listener can register to any number of emitters. Additionally,, it is also possible to stop a listener receiving events from a emitter. This is done by de-registering the listener via *emitter.remove_handlers(...)*. Not to forget the event propagation. An event is spread to all handlers from the top of the emitter stack until one returns EVENT_HANDLED.

Arianne



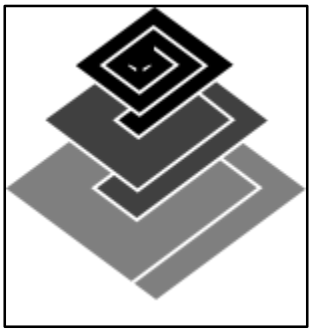
Arianne is an open source, multi-player online framework and engine to develop turn based and real time games. Its design provides a portable, robust and multithreaded server architecture, which is called “Marauroa”. It can be used to easily create games. Python is used for writing game rules, but Java is required for the server. Additionally, a MySQL backend powers the server and the server communicates with players using a TCP transport channel. The reference clients are coded using Java and C in order to achieve maximum portability. Moreover, Arianne’s server is totally client agnostic, which makes it very modifiable.

Since 1999, Arianne has been in development and its key concept has always been KISS: Keep it simple, stupid. It has evolved from a tiny application written in pseudo-C++ to a powerful, expandable server framework running on the Java platform and a portable client framework written in bare C to allow total portability of Arianne’s clients.

Marauroa

Marauroa is based on a kind of philosophy that is called Action/Perception. Each turn one or more clients can ask the server to do any action on their behalf using *action* requests. A *perception* is then sent to the clients in order to explain to them what they perceive of the world around them, and thus the result of their action. Like already stated above, Marauroa is totally game agnostic and makes very little assumptions about what you are trying to do with it. Because of this, it allows great freedom to create whatever type of game you want.

Twisted



https://upload.wikimedia.org/wikipedia/commons/f/f6/Twisted_Logo_%28software%29.svg

Twisted is an asynchronous event-driven networking engine written in Python and licensed under the open source MIT license. It runs on Python 2 and an ever growing subset also works with Python 3.

Twisted also supports many common network protocols, including SMTP, POP3, IMAP, SSHv2, and DNS and claims to make it easy to implement custom network applications.

The design aims to completely separate logical protocols and physical transport layers, the connection happens at the last possible moment.

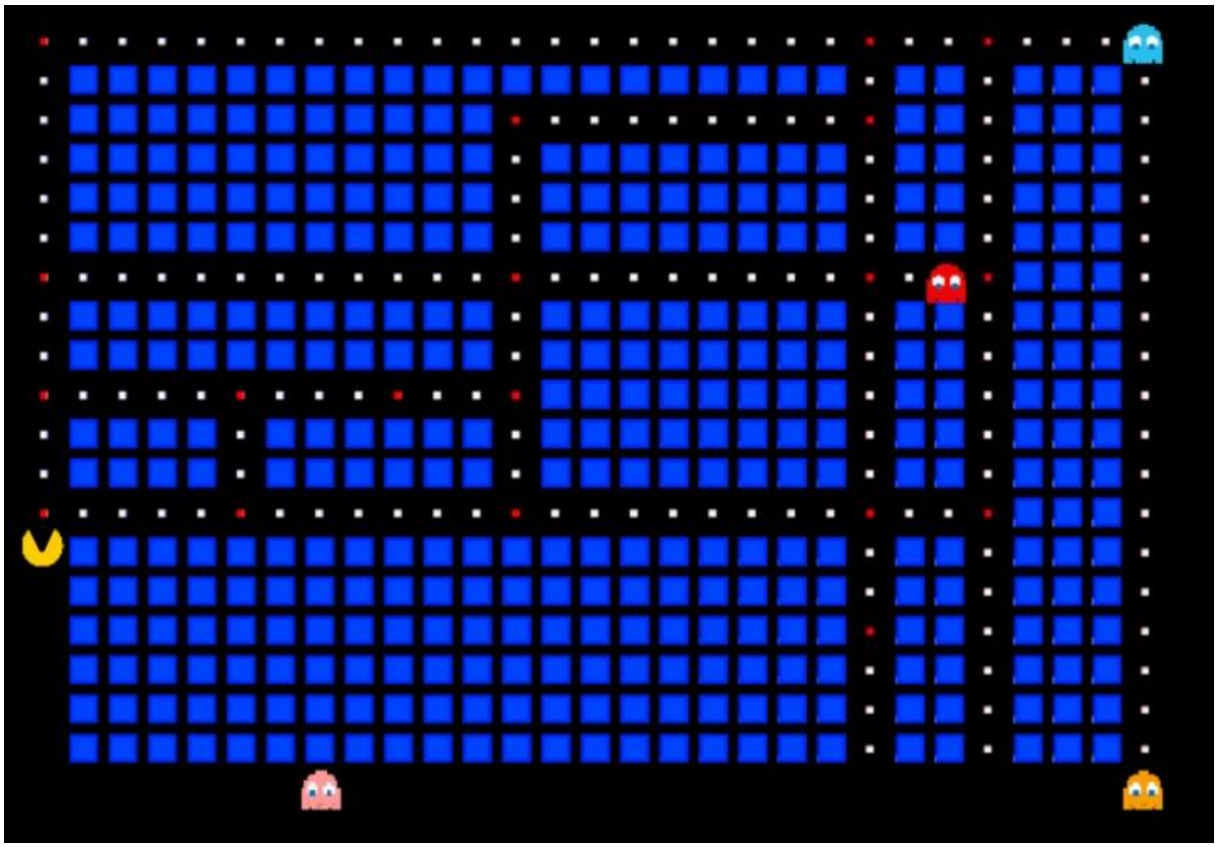
The framework can be used on the server-side as well as on the client-side. Its main component is the reactor which can be compared to an event loop on a node.js server. A ServerFactory is responsible for returning a protocol object whenever a client connects to the server - so each connection has its own instance of a protocol. The factory can be designed not to return a protocol instance in certain situations to reject the connection. On the client-side, the idea is similar: A ClientFactory returns a protocol object for every established connection.

A protocol basically has the Methods connectionMade(), connectionLost() and dataReceived() which are called on the applicable event and can be overwritten to perform appropriate reactions.

As mentioned above, Twisted is asynchronous. A central concept to implement this are so-called “deferreds”. A deferred can be returned like an ordinary object without delaying the programme while waiting for a response from the network or elsewhere. A deferred represents a response that is not there yet and acts as a promise that there will be a response sometime in the future. Methods that should be executed as soon as the response is available can be added to the deferred as callbacks (for data) or errbacks (for errors). A callback and an errback always appear in pairs. If only one of them is added to the deferred, a “passthrough” method will be defined and added automatically for the other.

Any number of callbacks and errbacks can be added to the deferred (“callback chain”) and will be executed one after another. If a callback or errback raises an exception or error, it is considered as failed and the next errback will be executed; if there is no exception, the next callback in the chain will be executed. Thus, error handling can be implemented in any link of the errback chain.

The Final Product



If you think, you have seen this image before, you could be right. Or it might just be a déjà vu.

Since we could not find out how to work with Arianne, we went back to our first attempt in the end, the cocos2d programme to be specific. The main functionality for a single player already worked, so we added some network functionality to it using the framework Twisted. Furthermore, we implemented a twisted server that would handle the communication between multiple players.

The server waits for clients to connect. In the beginning of the game it creates a random labyrinth to send to the clients so that every player uses the same one. In the further course of the game the server's main purpose is to forward command strings from one client to all others.

The client uses two threads to handle all needed tasks. The cocos2d director runs in the main thread, which is therefore responsible for the graphical user interface. The network communication is handled in another thread that runs as daemon in the background.

```
class networkThread(threading.Thread):
      def __init__(self):
            threading.Thread.__init__(self)

      def run(self):
```

```
            client.reactor.run()


Def main():

      thread = networkThread()
      thread.daemon = True
      thread.start()

      director.init(resizable=False, caption="HATman")
      game = GameScene()
      director.run(game)
```

A deferred object handles the strings that the client receives from the server. Via a callback method, the strings are written to a list in the appropriate layer of the character they refer to. The necessary methods are executed later (see below).

The GameScene class is derived from the cocos2d object Scene and contains different layers for the object that are needed in the game. The LabLayer contains the sprites for the labyrinth and some methods to handle the ways within it. Spots in consistent distances are defined as LabNode objects that can either be walls or ways. Every dead end or crossing is defined as a crossNode with at least one and at most four neighbour crossNodes. The nodes between two neighboured crossNodes are defined as wayNodes and determine whereto the characters can move.

The moveable objects (in concrete the pacman and the ghosts) derive from a CharLayer class that contains the variables and methods that are needed for all moveable objects, such as sprites, keyhandlers and a rectangle around them for collision detection. The GhostLayer and PacmanLayer class contain some additional variables and methods that are specific for the particular character.

Back to the GameScene. The methods that are necessary for the gameplay and require some sort of an interaction between the layers are located here. They are repeatedly called in the schedule method of the director. The method setDirection changes the direction of a character in consequence of a keypress if there is no wall standing in the way (quite literally). CheckBorders checks if a character reaches a dead end and forces it to stop in that case. Also the network communication is handled here. On every call of the schedule method, a string containing the current position of a character is sent to the server and received position updates for other characters are executed.

Conclusion

While it is definitely possible to write a multiplayer game in Python, it is not as easy as one might think at first. You have to know your way around different components and how they can interact. It takes a lot of research and trial by error to find the framework most suited for your endeavor and quite some time to get acquainted with it.

Therefore one should check beforehand if the given problem or a similar project has been solved already and if yes, what frameworks were used. This can save you a lot of trouble.

Also time management is of the essence with all projects and should not be neglected.

We learned a lot in the course of this project, some things concerning Python, namely that versions 2 and 3 and their subversions are not compatible, there are slight and not so slight differences and changes that make it impossible to make two different versions work together; also we learned anew how important line indentations are and to actually read error messages, even if they are annoying there might be something of importance hidden in there; and we got to know the framework Twisted, which is a mighty little piece of code that needs some getting used to but if handled correctly enables you to model network connections.

Ideas for the future

Due to the high amount of problems during our project and our lack of time management, we did not come to a consummate completion. There are still a lot of problems to fix and a lot of ideas to realise - as usual for software. There will always be an update.

Something that would not require any more programming effort, but make the game much more standing out, would be to draw own sprites that are different from the typical pacman design. Also some music would be nice.

Furthermore, we thought of a few little changes in the gameplay to make it more interesting. For example giving the ghosts some special abilities like walking through walls, locking ways or not being seen by the pacman all the time would be quite exciting.

On a more technical basis, a database could be added to save player accounts and high scores. This could also make it possible to play more than just one game on one server at the same time and coordinate the users like in a paper prototype we draw earlier: