Packman game

Denis Dubin
ČVUT–FIT
dubinden@cvut.cz

May 18, 2023

1 Disclaimer

This work was done with minimal use of third-party sources in order to avoid plagiarism. The algorithms may be inefficient, naive, mediocre or even talentless. The understanding of the game and its concepts is taken from childhood memories and may not match the information in mass use. Its architecture was not planned (this was intended) to create a proof of authorship for this work. The following will describe its writing process, functionality, the errors themselves and possible solutions.

2 Setting up the project and common info

The game was developed and tested on Windows 10 using the PyCharm IDE (was also tested on Linux Mint Ulyana), Python version 3.10. Multi-platform was not the goal of the project and, as a consequence, it can, but does not have to be compatible with other platforms. Due to the presence of all the relevant files generated by the IDE in the folder, it does not require anything the first time you run it with PyCharm. In case of launching it from the terminal you may need standard procedures for pygame, described on the website of the library developer.

3 Game concept

In the game pacman you move in two-dimensional space and have for the purpose to pass through each cell (collect the object which is located in each cell level) you are hampered by enemies who at collision with you kill you. On the level there is a buff that gives you a short period of time to kill your opponents and not to be killed yourself. Two more modifiers were also attached to the concept, speeding up and slowing down the main game cycle. You win if you collect all unique objects from each level cell. At the end you get a score of your session, which is based on the speed and number of killed opponents.

4 Color map

- Yellow – Yellow color for fruit
- White – White color for empty cell
- Green – Green color for player
- Red – Red color for enemy
- Blue – Blue color for wall
- Orange – Orange color for speed-up boost
- Purple – Purple color for speed-down boost
- Pink – Pink color for speed-down boost

5 Continued description of the game and the development process

5.1 Render

The first task, in my opinion, was to implement the rendering as it would help me analyze and debug the rest of the work.
After a not-so-long search for sprites, I came to the conclusion that the simplest option would be to fill a segment of the screen with a solid color. In the last project I had to spend several days searching for simple publicly available licensed sprites and several hours on the animation of the snake's head, and I had no desire to repeat this experience. After finishing the work I can say that probably the best way to implement the process of rendering would be to create an array of tasks, where each object in his change would throw the updated data and the use of two render buffers, then it would be enough to use the previous frame on top of which would be displayed only changed data (which in my case only a couple of objects)

5.2 Movement and collision

5.2.1 Begining

I started testing the movement with a single object in the form of a player and used the Pythagame API to synchronize the instances. Almost immediately it became obvious that it was necessary to tie the game to a clear time interval to create a specific tempo of the game. The collision is implemented in the character movement function by checking the type of object that is in the cell where the player is trying to go. This is probably the best way to implement collision, as writing an internal collision engine would be an overkill in a simple game like this. The implementation of collision in enemies is completely equivalent.

5.2.2 Enemies

The behavior of the enemies is extremely trivial. At the start they take one of the directions where there are no obstacles. Until there is no wall in front of them or the number of free directions is equal to two, they will continue to move in that direction. When one of the two rules is broken, they take a new randomly available direction and move again until one of the two rules is broken.

5.2.3 Communication between player and enemies

For a long period of time when the player and the enemy collided, the python would catch an error and crash. I put off solving this problem until the last minute because it was essentially very smooth at the end of the game. The kill implementation was done near the end of development with the movement feature modifications where I just made a bonus check, however, it still can happen when you are at the end of the corridor and walk through your opponent. This happens very rarely and even after conducting a lot of experiments and spending a solid amount of time on this problem, I could not figure out how to solve it.

5.2.4 Adds

At a further iteration of the development it became apparent that the bonuses need to be removed from the grid anywhere because of the possibility of enemies to pass through it. A separate class was created, which stores only coordinates and type of bonus. The array of objects of the class was also used for rendering in several passes, namely first the grid, then the bonuses.

6 Bonuses

For bonuses, a separate timer was added to the game loop to keep track of the duration of bonuses. It also kept its type.

6.1 Speed bonuses

The speed bonuses were mainly to the time interval, or rather to its modification. All internal operations are in the ife which is played only when the timer has passed a certain amount of time. It was enough just to bring it to a variable, which is updated through the type of bonus and keep a second timer, which tracked the duration of the bonus and returned the modifier to the standard value

6.2 Hulk bonus

It was impossible to do without changing the logic of movement on both sides. Specifically, we should have added a check when attempting to kill on the type of current bonus. Accordingly, if it is present, remove the enemy and update the renderew.

7 Finishing and Game score

The game ends when the last fruit is left on the grid. The overall scoring is done by a formula that clearly corresponds conceptually to the original game, but probably differs afterwards in terms of element coefficients.

8 Map and map loading

For maps, a fairly simple boater was created that fills the grid directly based on the line and the position in the line on the symbol. It does not test or check the readable map, which in its own way gives free rein to imagination and space for mind games, but I was too lazy to make a map in the form of a star or sickle. If you have such a desire, you can check out the load_map function. It is readable enough to understand which symbol has what meaning.

9 Conclusion

Most of the conclusions have been described above and have already earned their mention. I can only add that it was necessary to add a different format of obstraction and architecture, namely the implementation of the classes of enemies and character.

References

[1] Python Software Foundation. Python documentation. https://docs.python.org/3/.

[2] Pete Shinners. Pygame documentation. online. https://www.pygame.org/docs/tut/PygameIntro.html.