PRIMEVAL MINI GAMES USING PYTHON

Purushotham Muniganti.
Professor, Dept. of Computer Science and Engineering,
Sreyas Institute of Engineering and Technology, Nagole,Hyderabad.
Email: purushotham.m@sreyas.ac.in

Abstract:

The first ever video game was developed in the year 1958 (often said to be *Tennis for two*). There are many genres under video games (example, role-playing games). Board games have been played as a part of most cultures and religions throughout history (Senet from 3300-2700BC of Egypt is said to be the oldest board game known to have ever existed). For this paper, our aim is to digitize and optimize such abandoned board games and primordial video games using state-of-the-art programming languages (python) to make them available for today's users. For this paper, we have developed 'snakes and ladders' (originally known as Moksha patam - ancient Indian game), optimized 'minesweeper' (single player puzzle game from 1960's) and 'snake game' (first developed and published in 1976) using the most basic features from python, the resulting system is fast and reliable.

Keywords: *Python, Tkinter, Snakes and ladders, Random, Minesweeper, Snake game.*

Introduction

Major stand-against faced while development of this paper is the ambiguity in information available across the resources and evolution of programming languages. There were numerous attempts by several people over the course of history to develop an out-and-out application as a compilation of such mini games [4], [8]. Although none of them were utterly successful as the technologies were ameliorated over the course of time and their work is no longer compatible with current versions. Example, Snakes and Ladders developed using Pygame (a feature from python

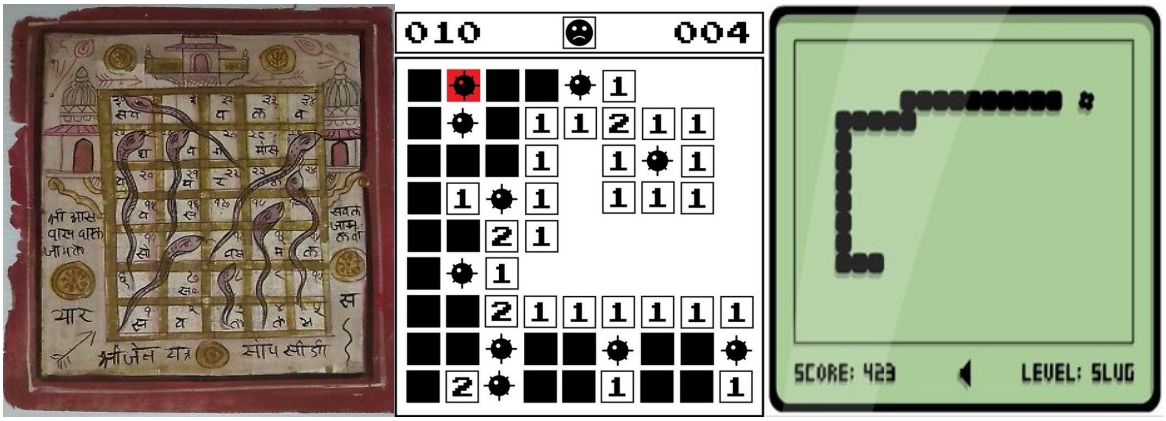
2.x - year 2000)) is not compatible with 3.x versions of python as Pygame module has been amputated. For this paper, we are using the most basic features of Python (Tkinter) to provide the Graphical user interfaces for our core (actual program) making it simple, reusable, reliable and compatible with adequate performance.

1. Applications

As we all know video games have been in swing since the day they were first developed[7];[8]. For this paper we are bringing back some of the obliterated games in a more approachable way by digitalizing the basic and optimizing the forgotten as these games also hold affective ties with many people. This paper will address our work as one stop solution for all the issues (old / non compatible programs spread all across the web) while providing reliability and performance.

1.1. Problem statement

The major problem faced today is amelioration of programming languages, leaving all the old work non-compatible [1], [3], [7]. By 2020 most of the versions of programming languages will be in EOL - end of life and receive no further official support, After that date there will be no further updates or Bug fixes (example, Python 2.x).All the old work will no longer be provided with any security, most of the modules will no longer be compatible and rest have relatively low yield. We also do not have a proper platform that provides users with all the mini games at one stop. The pictures beneath serve as existing systems for the mini games,



(a) (b) (c)

Fig 1.1: (a) Hand drawn board of snakes, ladders (b) Minesweeper (c) Snake game from Nokia 3310.

2. Related work

There is a lot of unseen work behind the development of these games [1][5] and[6], They are developed in such a way that they perform adequately while having just the basic methods (for snakes and ladders we are providing virtual, pixel based board to the program while having an image of a similar board in the background [9]. we can change/remove the image and program will still function the same way).

2.1 GUI

We are using TKinter to provide graphical user interface to our core (games) [1]. Tkinter is Python binding to the Tk GUI toolkit, is written by Fredrik Lundh. It is free software under the Python license. Tkinter is incorporated as 'Themed Tk' in python 2.x and 3.x, this allows Tk widgets to be themed to look like native desktop environment, is included with Linux, Microsoft Windows and Mac OS X installs of python[2]. This provides our program with compatibility against any of these operating systems/ environments with minimum changes.

2.2 Dice roll + mine/obstacle generation

We are using module: random for most of the features provided in our paper. This module contains a handful of functions, it can select a random item from a list, rearrange the list in random order, and select a random number from given range. Our program (snakes and ladders) selects a random number between 1 and 6 and displays it every time the user clicks on 'roll' button. This function imitates unfeigned dice roll. Similarly for minesweeper and snake game, mines/Obstacle are randomly generated by selecting a random (row, column) combination. This will dispense users with a unique experience each time.

3. Approach

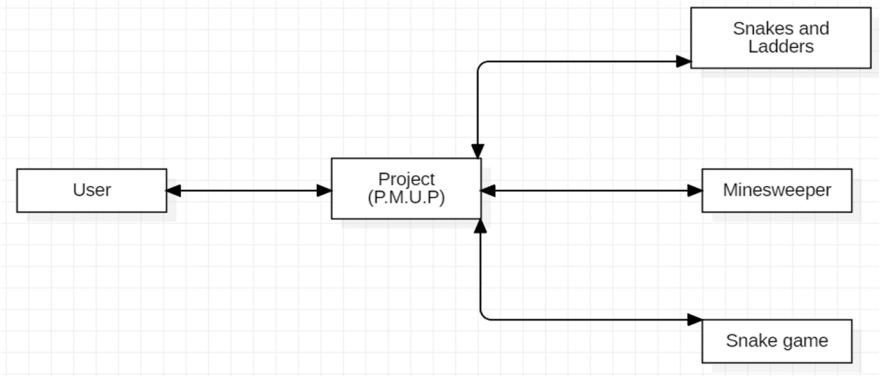


Fig 3: The basic architecture of Our Model

3.1 Snakes and ladders

Our approach towards developing snakes and ladders is based on Dimensions and pixel count. We are simulating dice roll by selecting a random number between (1, 6) using ‘random’ method of python. Snakes and ladders in the game are pre-defined based on the image (using pixel count, when player piece reaches X return to Y). Once we fire up the game, list box with options up to 6 players is displayed, as soon as the user selects number of players(n) and hits the ‘start’ button, n player pieces are displayed on the board, A box to display dice value and ‘roll dice’ button pops up on the canvas. Players can roll dice again each time they take a ladder or whenever they are bitten by a snake. First player (piece) to reach 30 will be shown as the winner.

3.2 Minesweeper

For this paper, we have optimized the minesweeper by generating random mines using random selection of row and column combinations on the board of size n x n instead of a predefined set. We are providing players with three levels of difficulty to choose from (easy [board = 10x10, mines=10], moderate [board = 15x15, mines=15] and difficult [board = 20x20, mines=20]), which in return will affect the board size and number of mines to be generated. We are using a recursive function to clear out as many blocks as possible (only when the block is a non-mine and if there are no mines in 8 directions of the block).

3.3 Snake game

Although the concept of snake-based games originated in 1976, there was a resurgence of interest only after a variant was preloaded on Nokia mobiles in 1998. We have optimized the exact same variant using Python programming language. When we fire-up the program, an empty canvas with 3 buttons (Start, Stop, and Quit) on its left is displayed. When user hits ‘start’ a snake is generated at predefined position on the canvas and an obstacle is generated at a random position. We can control snake movements using “UP”,”DOWN”,”RIGHT” and ”LEFT” keys of our keyboard. When head of snake collides with the obstacle, the Obstacle disappears, snake size increases by one block (0.9) and a new obstacle is generated at some random position on canvas other than the areas covered by snake parts.

4. Experiment results

4.1 Result for snakes and ladders

As experiment results for snakes and ladders we have a fully developed game module that accommodates up to 6 players with simple Graphical user interface from Tkinter. Player pieces on the canvas move according to the generated dice value each time a player hits 'roll' button.

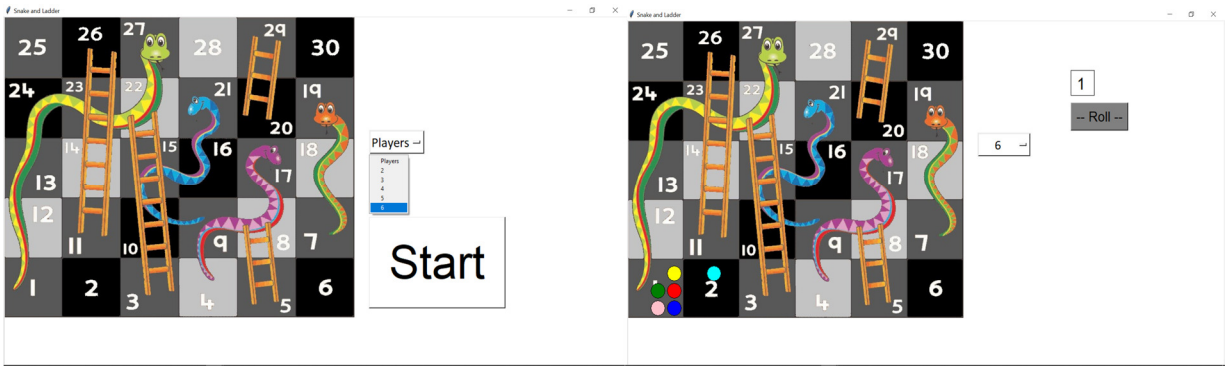


Fig 4.1: Graphical User Interface for 6 Players.

4.2 Result for minesweeper

As the experiment result for our Minesweeper module, we have fully optimized game with three difficulty levels for the user to choose from. Mines on the board are random generated and blocks are cleared using recursive function, we can also flag a block by right clicking on it, flag count is displayed at the bottom.

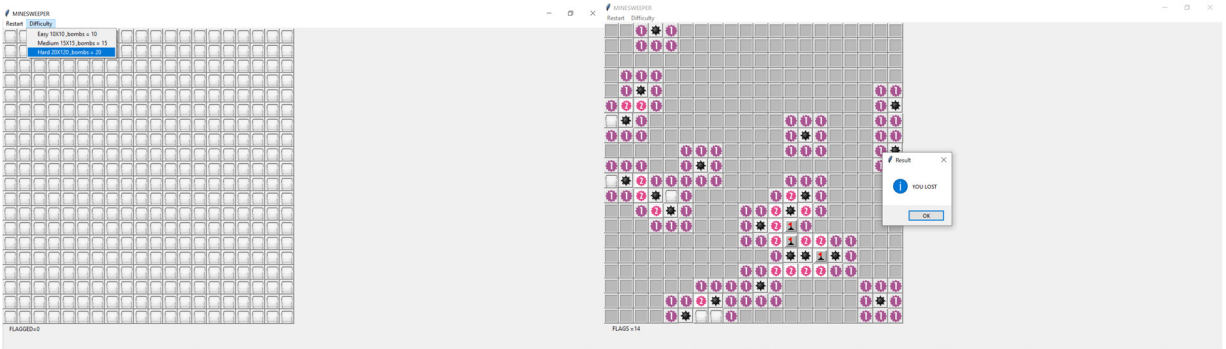


Fig 4.2: Experimental Results for Minesweeper Module.

4.3 Result for Snake game

As the experiment result for snake game, we have fully optimized game that generates the obstacles randomly across the canvas anywhere other than the blocks that are occupied by the snake body parts. Even after firing up the module, it is under halt state until user clicks on the ‘start’ button. The game stops when user clicks on ‘stop’ button and the window is shut close when user clicks on ‘quit’.

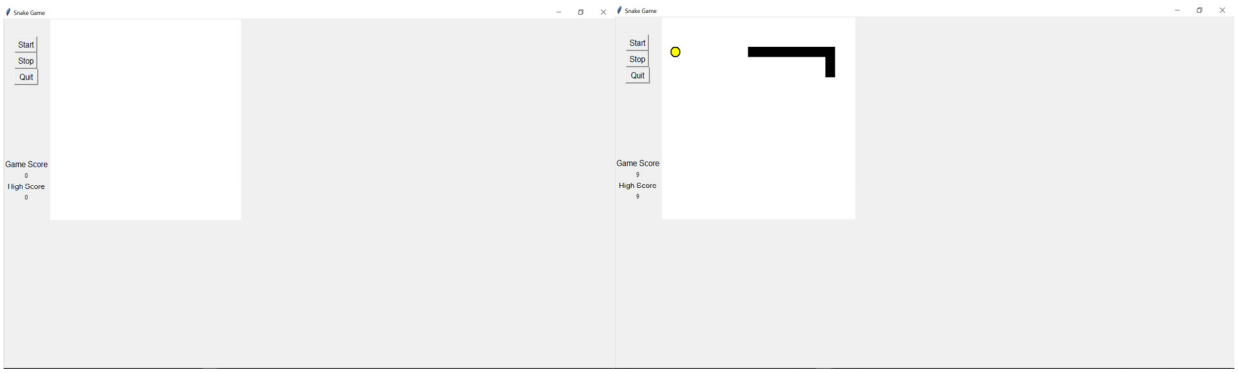


Fig 4.3: Experimental Results for Snake game with obstacles.

5. Conclusion

We concluded that our results serve as one stop solution for people who prefer long-in-the-tooth type of games. Our aim is to develop/optimize the classics without having them lose their essence while providing a window for new developments and improvements. As our idea for the future, we can further improve by using multithreading concepts (as python does not support double import) and develop a real-time, web-based application with many other primeval mini games.

6. References

[1] Shipman, John W, *Tkinter reference: a GUI for Python*, New Mexico Tech Computer Center, retrieved 2012-01-11
[2] Klein, Bernd. "GUI Programming with Python: Events and Binds". *www.python-course.eu*. Retrieved 18 August 2018.
[3] Topsfield, Andrew. *The art of play. Board and card games of India*. (2006).
[4] Elliott Avedon Museum & Archive of Games. “Snakes and Ladders". Archived from the original on 2008-02-20.
[5] Slesin, Suzanne. At 50, Still Climbing, Still Sliding New York Times 15 July 1993

[6] Adamatzky, Andrew "How cellular automaton plays Minesweeper". *Applied Mathematics and Computation*. (1997).
[7] Mordechai Ben-Ari Minesweeper is NP-Complete (Report). Weizmann Institute of Science, Department of Science Teaching. — An open-access paper explaining Kaye's NP-completeness result (2018).
[8] Rusel DeMaria & Johnny L. Wilson. *High score!: the illustrated history of electronic games* (2003).
[9] Neon Cherry. "The Unofficial Nokia Gaming Blog ". Web.archive.org. Archived from the original on November 1, 2007. Retrieved November 4, 2011.