PPC project

Hannabis

Razmig Kéchichian and Tristan Roussillon

1. Goal

The goal of this programming project is to design and implement a multi-process and multi-thread multi-player game in Python. This document defines the functionalities that you must implement at a minimum. Some questions are deliberately left open for you to propose your own solution. Any extensions and developments will act in your favor in the evaluation. Be rigorous and creative!

2. Presentation

2.1. Minimum specifications

Hannabis is a simplified version of the Hanabi cooperative card game. Its deck contains numbered cards in as many colors as there are players (e.g. for 2 players there are 2 colors: red and blue, for 3 players there are 3 colors: red, blue and green, etc.): three 1s, two each of 2s, 3s, and 4s, and one 5. The game begins with `number_of_players + 3` information tokens and 3 fuse tokens. Players are dealt a hand of five cards. They can see each other's cards but not their own. Each turn, a player must take one of the following actions:

- give information: pointing out cards of either a given number or a given color in the hand of another player (e.g. "This card is your only red card," "These two cards are your only 2s", etc). Giving information consumes one information token.
- play a card: by choosing a card from the hand and attempting to add it to the cards already played. This is successful if the card is a 1 in a color suit that has not yet been played, or if it is the next number in a suit that has been played. Otherwise a fuse token is consumed and the misplayed card is discarded. Successfully playing a 5 of any suit restores one information token. Whether the play was successful or not, the player draws a replacement card from the deck.

The game ends when either all fuse tokens are used up, resulting in a game loss, or all 5s have been played successfully, leading to a game win.

2.2. A minimal design

Hannabis involves 2 types of processes at least:

- `game`: implements the game session, manages the deck and keeps track of suits in construction
- `player`: interacts with the user [1], the `game` process and other `player` processes keeping track of and displaying hands and associated information. Interactions with the `game` process are carried out in a separate thread.

Inter-process communication: Suites in construction and tokens are stored in a shared memory accessible to all processes, `player` processes communicate with the `game` process via sockets and among themselves using a message queue requiring a carefully designed exchange protocol, end of game events are notified via signals emitted by the appropriate process in each situation.

3. Implementation

3.1. Design

Start with a diagram to better visualize the interactions between processes/threads. State machine diagrams can be very helpful during this stage. The following points need to be addressed and justified:

- relationships between processes (parent-child or unrelated)
- message queues used in communication and messages exchanged between processes along with their types, content and order of exchange

1 If necessary, refer to this Stackoverflow thread for non-blocking console input solutions in Python. Also, if separate physical terminals are desirable for players, consider executing the player program remotely via SSH.

- sockets used in communication between processes along with exchanged messages, their content and order of exchange
- data structures stored in shared memory and how they are accessed
- signals exchanged between processes along with their types and associated handlers
- synchronization primitives to protect access to shared resources or to count resources
- tubes involved in pairwise process communication, if any, and their types

Write down a Python-like pseudo-code of main algorithms and data structures for each process/thread to help you in implementation.

3.2. Implementation plan

Plan your implementation and test steps. We strongly encourage you to first implement and test every process/thread separately on hard-coded data in the beginning. Then you can implement and test each pair of communicating processes/threads, again on hard-coded data in the beginning, if necessary. Only after implementing and testing each inter-process communication you can put all processes together and test the game. Take care of the startup and the proper shutdown of the game, freeing all resources, making sure you are able to demonstrate end of game scenarios during a short presentation. Identify possible failure scenarios and think about recovery or termination strategies.

4. Deadlines and deliverables

15/1/2024	project proposal is published on Moodle
30/1/2024 - midnight	submission of project code (including a README explaining how to execute it) and report archive on Moodle, refer to organization slides for the content of the report
31/1/2023	15-minute demonstration per project with the tutor in charge of your group