**PPC project Hannabis**

1. **Conception**

**Structure of program**

* Parent Process: Game Process

It manages the game state, such as card stacks, scores, information tokens and fuse tokens.

* Child Processes: Player Processes

Each player is a process, these processes are side-by-side, communicating with the parent process (the game process) via a socket, and each player process communicates with each other via a queue.

* Multi-threading: There are three threads in each player process, the first one is used to handle the game logic, the second one is used to handle the communication with the parent process, and the third one is used to handle the communication between players.

**Explanations of choices**

Modularity and clear division of responsibilities: Separating the game logic, communication with the parent process, and inter-player communication into different parts allows each part of player process to have a clear and dedicated responsibility. This helps with code organisation, maintenance and debugging.

Parallel threading: Using multiple threads allows different tasks to be handled simultaneously. For example, one thread can handle game logic while another thread waits for a message from the parent process or another player. This can improve the responsiveness and efficiency of the program.

Flexible communication mechanism: By using sockets and queues, this design provides a flexible communication mechanism. Socket communication between parent and child processes is suitable for infrequent but important game state updates, while queue communication between child processes is suitable for frequent and fast information exchange.

Improved scalability: Each player runs as a separate process, which increases the scalability of the system. It is easier to add more players or adjust the rules of the game.

Fault tolerance and stability: Isolating players at the process level means that the failure of one process does not directly affect other processes. This is useful for maintaining the stability of the entire game.

1. **Implementation plan**

**Important pseudo-code**

**Game Process:**

initialiseCards( )

distributeCards( )

while not gameOver:

for player in players:

requestAction(player)

action = listenToPlayer(player)

result = updateGameState(action)

sendUpdateToPlayer(player, result)

if checkGameOver( ):

gameOver = True

break

endGame( )

**Player Process:**

# Thread1 - Game Logic

Thread1\_gameLogic:

while not gameOver:

action = getUserInput( )

validateAction(action)

sendActionToGameProcess(action)

update = getUpdateFromGameProcess( )

displayUpdate(update)

# Thread2 - Communication with Game Process

Thread2\_communicationGame:

while not gameOver:

message = receive.socket( )

processGameMessage(message)

# Thread3 - Communication with Other Players

Thread3\_communicationPlayer:

while not gameOver:

playerMessage = receive.queue( )

processPlayerMessage(playerMessage)