Discussion

Explanation:

The solution asked for the following changes to the game:

1. A player can make a move up to once every two-seconds. Once the player moves, their colour changes to a different hue to indicate they are stuck.
2. In the following two-seconds, all the other characters make their moves "in parallel". Characters still have a variable number of moves, those moves are made in the time available (i.e., faster if there are more moves to be made).
3. Once that two-seconds is up, the player is able to move again and their colour changes back to normal.
4. The game waits for the player to make their move.

The solutions are implemented in the game in the following way:

Change 1 is implemented using the runnable interface in the Player Class. New attributes thread and threadSleepTime have been added in lines 13 and 14 to keep track of threads and time the current player thread should sleep for. The notify method has been modified to update the global variables char c and GameBoard<Cell> gb which were earlier parameters to the notify method in player. This was necessary in order to make them available to the run() method in player as run() does not take any parameters. Then a new thread is started inside the notify method each time it is called given the previous thread it started has terminated or thread is null, which is when notify is invoked for the first time. Notify then starts the thread by running thread.start() which then invokes the run() method. The run() has only one method call inside it movePlayer(c, gb). The movePlayer method is as it was before except it has been put inside a try and catch block to catch the InterruptedException from thread.sleep(). There is a new method updatePaint() in player which changes the player colour to red and invokes the paint() method once the player has made its move. After that thread.sleep(threadSleepTime) is executed on line 62 which makes the thread sleep for threadSleepTime which is 2 seconds or 2000 milliseconds. This sleep method call makes the player thread sleep until the other characters have made their moves.

Change 2 has been implemented using a runnable interface in the Character class. New attributes Stage stage and Thread thread are added on line 23 and 24 in the character class in order to keep track of threads and make the stage available to the makeMove() method inside run(). The aiMove() has been modified to update the global attribute stage every time the method is invoked. This method also creates a new thread given the previous thread it created has terminated or it is null which is the case when it is invoked for the first time. The thread.start() method is then invoked which invokes the run() method and which in turn invokes the makeMove() method for each character. Method makeMove() chooses the right behaviour for each character and performs the move.

Change 3 and 4 have also been implemented inside the movePlayer method of Player which invokes the updatePaint(). After the thread wakes after a set sleep() time of 2000 milliseconds or 2 seconds, the updatePaint() method is called again to change the Player's colour back to orange to show that its ready to make a move. The player can now make a move as a new thread will be created inside the notify method of player as soon as a key is pressed.

This particular design solution has been chosen because it is easy to implement, not too complicated and executes the program in the desired way. It creates a new thread for each move and then terminates the thread as soon as the move is done. This prevents the complexities of running a while loop inside the run method to keep the thread running as long as the game is still running and the characters are waiting for their moves. The solution also prevents issues with inconsistency and thread safety as it creates a thread for new move and then terminates that thread after the move is done. It implements the threads using the runnable interface which is a simpler solution than the other more complicated alternatives like extending the Thread class.

Discussion

The only problem encountered with the solution above is to make the Character threads execute their moves with a relative speed. The idea was to put the character threads to sleep for 2000/ movesLeft milliseconds after executing each move which would have maintained the speed of the moves of each character accordingly to be executed within the 2000 milliseconds that the player thread is sleeping for. However, this implementation of making the character threads sleep after each move, created a thread inconsistency with the sleep() method causing the GUI to freeze. Therefore, the characters now make their moves as soon as the player thread makes its move without a relative speed. However, a little delay has been added in the main() method to replicate this thread.sleep() by adding an if construct to update the stage by running stage.update() every 500 milliseconds instead, which is done inside the if block on line 24 of the main method().

Alternative solution:

An alternative solution to this task could have been to implement the thread structure by inheriting the Player and Character class from the Thread class. This alternative would have created a new thread for the player and each character when they are instantiated. However, this would have needed a while loop inside the run() method of each thread to keep the thread running as threads in Java terminate once the run() method is executed. The individual character threads would then have waited for the player thread to make a move by running the wait() method inside the while loop until after they are notified by the player thread that the player has made a move using the notify() method. Without the while loop the thread would terminate and all the moves would load onto the AWT-EventQueue thread causing inconsistent behaviour such as GUI freeze. This alternative solution would also have created more issues relating to consistency of objects and added additional complexities to maintain thread safe behaviour.