JOCS DE PROVES

TEST1: t1_rules.json t1_board.json | t11_log.txt , t12_log.txt , t13_log.txt

Per el primer joc de proves, utilitzarem la versió proporcionada per el anunciat del projecte. Aquesta ens dona una versió del joc juga-ble amb un numero normal de caselles repartides entre tots els diferents tipus. Això ens permetrà comprovar el funcionament base del joc i trobar els possibles errors principals o modificacions necessàries per poder jugar sense entrebancs i amb diàlegs amigables.

Aquest joc de proves farem tres execucions. La primera serà una partida entre dos usuaris (en aquest cas nosaltres jugarem amb els dos). En aquesta execució provarem totes les funcions principals del programa (el moviment en el taulell, la resposta quan cau en una casella, les execucions de les accions opcionals, les diferents accions de cada casella, la compra de terreny, edificació de apartaments i hotels, entre altres).

En aquesta execució ens hem trobat amb uns bugs molt simples, casi tots els bugs de son de sortida d'informació per pantalla o en el fitxer de desenvolupament de partida. El primer era la repetició de la descripció de la casella sort a la que un jugador queia

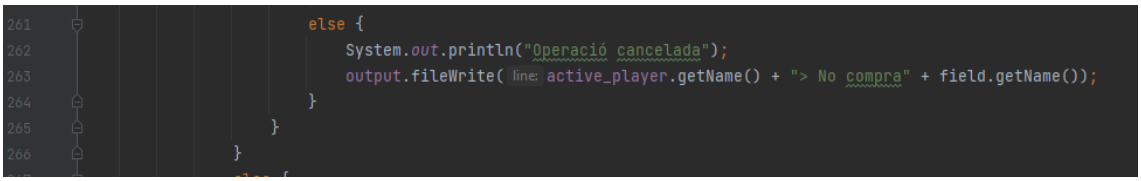
```
--------------------------------------------------------------------

Has caigut en una casella de sort
Vas immediatament a la casella 1 i si passes per la casella de sortida, cobra la recompensa
Vas immediatament a la casella 1 i si passes per la casella de sortida, cobra la recompensa
1 - 10000
2 - terreny
```

Per solucionar-ho hem borrat la sortida per pantalla repetida en algunes funcions d'execute de les cartes, en aquest cas podem veure que el error el trobaríem a la casella sort amb la acció de "CardGO".

El següent error que ens hem trobat ha estat gràcies al fitxer de log generat per la partida. Podem veure com encara que compréssim un terreny, ens donava una sortida de terreny no comprat.

```
CANVI DE JUGADOR:
Usuari1> 40000€, Terrenys 0, Prestecs 0, Posició 1
Usuari1> Resultat dels daus: 3 i 4
Usuari1> Mou de la posició 1 a la 8
Usuari1> Cau a Casella Terreny
Usuari1> Compra CarrerCamprodon a un preu de 8000
Usuari1> No compraCarrerCamprodon
Usuari1> Accio opcional escollida: 1 - OpActSell
Usuari1> Ven CarrerCamprodon per 8000€
        Usuari2> Ofereix 10000€
Usuari2> Compra CarrerCamprodon a Usuari1 per 10000
```

Per sol·lucionar-ho hem posat el output.fileWrite (funció que escriu al log) dins del else ja que sino s'executava sempre.



Per últim hem vist que el fitxer de log no mostrava la informació quan la venta (Sell) es cancel·lava ni la informació de opActLoan(Acció opcional lloguer) ni opActLuck (Acció opcional sort).

```
CANVI DE JUGADOR:
Usuari2> 40000€, Terrenys 0, Prestecs 0, Posició 7
Usuari2> Resultat dels daus: 2 i 2
Usuari2> Mou de la posició 7 a la 11
Usuari2> Cau a Casella Sort
Usuari2> Vas immediatament a la casella 1 i si passes per la
Usuari2> Ha rebut: ParcDelMigdia
Usuari2> Accio opcional escollida: 1 - OpActSell

CANVI DE JUGADOR:
Usuari1> 5000€, Terrenys 2, Prestecs 0, Posició 17
Usuari1> Resultat dels daus: 1 i 3
Usuari1> Mou de la posició 17 a la 1
Usuari1> Ha rebut: 10000€
Usuari1> Cau a Casella Sortida
Usuari1> Accio opcional escollida: 2 - OpActLoan
```

Per solucionar-ho hem posat els outputs fileWriter() en les funcions corresponents dels executes ja que quan cancel·làvem, no escrivíem res en el fitxer de log.

Un cop feta la execució entre Usuari i Usuari, realitzarem una segona prova entre un Usuari i la CPU. Això ens proporcionarà una visió mes detallada de les accions de la CPU ja que veurem a temps real cada moviment. També podrem veure possibles errors de crides a la CPU, respostes il·lògiques de aquesta i errors ens sortides de dades.

En aquesta execució no hem trobat cap error. Cal dir que la execució aquesta s'ha realitzat posteriorment a els testos que venen a continuació, i per tant la majoria de errors ja han estat solucionats per jocs de proves mes precisos.

Per la tercera i última execució de aquest joc de proves, realitzarem la prova amb 4 CPU. Axis podrem veure com realitzen les operacions de subhasta, negociacions, i decisions de la CPU en general. El joc funciona correctament i la CPU també. Les decisions principals les realitza de forma raonablement lògica i això l'hi permet poder realitzar una partida sencera sense cap problema.

Hem trobat però, que la CPU al caure en fallida intentava vendre les propietats i la partida era pràcticament impossible de acabar. Llavors hem canviat perquè la CPU es declares sempre en fallida. Això ens proporciona la oportunitat de provar quan acaba la partida no per màxim de diners o torns.

TEST2: t2_rules.json t2_board.json | t2_log.txt

En aquest segon joc de proves comprovarem el funcionament del programa quan tenim una poca quantitat de caselles. En aquest cas, utilitzarem totes les caselles del tipus sort i així, comprovem també possibles errors en aquestes. El taulell doncs, estarà format per 4 caselles. La primera de Sortida i 3 caselles de tipus sort.

Al inicial el joc de proves, ens donem compte de que hi ha un petit error en el moviment del jugador, ja que quan aquest arriba al final de la taula, reinicia el pas per aquesta des de la casella 0. Una casella inexistent en el taulell ja que està organitzat amb una primera casella null, i llavors les següents caselles de la 1 al final. Amb això canviat, procedim en la execució.

També veiem que, per el contrari que en el joc de proves primer, aquest te menys de 12 caselles i existeix una possibilitat no contemplada inicialment de que amb un resultat dels daus el jugador passi varies vagades per la casella de sortida. Per solucionar-ho realitzem un petit canvi que cicli per totes aquestes passades per la casella de sortida i calculi la posició final correctament.

També, per seguretat canviem la funció de execució de la carta de tipus Anar (CardGo) ja que aquesta no tenia en compte internament si se l'hi passava un valor superior al nombre de caselles del taulell. Per solucionar axiò realitzem el càlcul del mòdul de la posició on ha de anar per el total de elements del taulell. Per exemple, en aquest cas tenim una carta que ens fa anar a la casella 17 i calcularem el mòdul sobre 4 (17 % 4 = 1) i ens dona la posició relativa on anirem.

Per últim, ens trobem que en la sortida de text i en el log, el usuari no està informat del tipus de carta de sort que ha trobat abans de haver de decidir si la vol guardar o no. Llavors, canviem el ordre de execució perquè tingui un sentit mes racional. Amb això fet, ens trobem, amb un altre problema, ja que al treure el missatge abans de executar-lo a que aquest estigui en estat null. Aixo es degut a que el missatge de la carta s'assigna abans de executar-la. Canviem llavors la assignació del missatge a el constructor i ens soluciona el problema.

```
CANVI DE JUGADOR:
CPU3> 37500€, Terrenys 0, Prestecs 0, Posició 4
CPU3> Resultat dels daus: 1 i 3
CPU3> Ha rebut: 10000€
CPU3> Mou de la posició 4 a la 4
CPU3> Cau a Casella Sort
CPU3> DEVOLUCIÓN DE HACIENDA COBRA 2000 €
CPU3> Accio opcional escollida: 2 - OpActBuy
CPU3> Cap jugador amb propietats
```

Resultat final de la crida de una casella de sort

TEST3: t3_rules.json t3_board.json | t3_log.txt

En el tercer joc de proves, comprovem el funcionament del programa amb nomes terrenys com a caselles (i la sortida en la posició 1). Això ens permet comprovar les funcions de compra, construcció i edificació sobre un terreny entre altres possibles problemes de execució i de sortides per pantalla. També, assignem nomes la opció de terreny com a recompensa així podrem realitzar comprovacions sobre aquesta acció.

Ens salta un error ja que al tenir nomes un element de recompensa no comprova si hi ha algun terreny per donar i quan no existeix el busca igual. Per solucionar-ho afegim una condició prèvia.

```
if(reward.equals("terreny")) {   reward: "terreny"
    BoxField field_reward = randomField();   field_reward: null
    if (field_reward == null) {   field_reward: null
        System.out.println("NO HI HA CAP TERRENY");
        output.fileWrite( line: player.getName() + "> No hi ha cap terreny");   player: CPUPlay
    }
    else {
        player.addBox(field_reward);
        field_reward.buy(player);
        System.out.println("Has rebut " + field_reward.getName());
        output.fileWrite( line: player.getName() + "> Ha rebut: " + field_reward.getName());
    }
}
else{
    player.charge(Integer.parseInt(reward));
    System.out.println("Has rebut "+reward+"€");
    output.fileWrite( line: player.getName() + "> Ha rebut: "+reward+"€");
}
```

Situació del error dins del codi

Ens trobem també amb un problema amb la funció de fallida (bancarrota) ja que dels tres jugadors, dos tot i estar a 0e segueixen jugant amb normalitat. Això es degut a un problema amb la lògica de retorn del booleà de la funció de fallida.

```
CANVI DE JUGADOR:
CPU3> 157000€, Terrenys 4, Prestecs 0, Posició 2
CPU3> Resultat dels daus: 4 i 1
CPU3> No hi ha cap terreny
CPU3> Mou de la posició 2 a la 2
CPU3> Cau a Casella Terreny
CPU3> Accio opcional escollida: 2 - OpActBuy
CPU3> Cap jugador amb propietats

CANVI DE JUGADOR:
CPU1> 0€, Terrenys 0, Prestecs 0, Posició 5
CPU1> Resultat dels daus: 5 i 1
CPU1> No hi ha cap terreny
```

El jugador CPU1 a 0€ i segueix jugant

```
} else {
    if (!board.isBankrupt(active_player,field.getRent(), aux: this)) {
        output.fileWrite( line: active_player.getName() + "> No pot pagar el llogue
        board.transferProperties(active_player,field.getOwner(), movement: this);
        active_player.goToBankruptcy();
    }
}
```

Solució: Eliminar el negat del if ja que si retorna true vol dir que ha entrat en fallida

El seguent problema que ens trobem es amb la mateixa funció de fallida. En aquest cas ha entrat en un bucle infinit.

```
CPU2> Cau a Casella Terreny
CPU2> Accio opcional escollida: 2 - OpActBuy

CANVI DE JUGADOR:
CPU3> 28658€, Terrenys 1, Prestecs 1, Posició 3
CPU3> Resultat dels daus: 5 i 4
CPU3> No hi ha cap terreny
CPU3> No hi ha cap terreny
CPU3> Mou de la posició 2 a la 2
CPU3> Es declara en fallida
CPU3> Es declara en fallida
CPU3> Es declara en fallida
CPU3> Es declara en fallida
CPU3> Es declara en fallida
```

Això s'ha solucionat modificant la forma en la que la funció retorna el valor i en comptes de guardar-lo en una variable, en el moment que sabem el resultat el retornem directament. Així trencant el bucle ja que aquest està declarat que acaba quan alguna de les accions de venta o ús de una carta sort s'ha executat correctament.

```
while (!sell_action_done && !card_action_done) {
    Scanner scan = new Scanner(System.in);
    int missing_money = pay_amount - current_player.getMoney();
    int option_nr = selectOption(current_player,pay_amount, missing_money,scan);
    if (option_nr == 1) {
        while (!sell_action_done) {
            sell_action_done = tryToSellToAlive(current_player,aux);
        }
    }
    else if (option_nr == 2) {
        card_action_done = tryToRunCardToAlive(current_player,missing_money,scan);
    }
    else {
        System.out.println("El jugador " + current_player.getName() + "s'ha declarat en fallida");
        output.fileWrite( line: current_player.getName() + "> Es declara en fallida");
        current_player.goToBankruptcy();
        return true;
    }
}
```

Solució del problema

Per últim, ens trobem amb un error que comporta tres conseqüències i errors relacionats. Ens trobem amb un dels jugadors jugant amb quantitat de diners negativa.

```
CANVI DE JUGADOR:
CPU1> -16404€, Terrenys 0, Prestecs 0, Posició 2
CPU1> Resultat dels daus: 1 i 3
CPU1> No hi ha cap terreny
CPU1> Mou de la posició 1 a la 1
CPU1> Cau a Casella Sortida
CPU1> Accio opcional escollida: 0 - RES
```

Primer error: la funció payRent, en cas de que despres de aconseguir els diners en la acció al entrar en bancarrota, aquest pagament no es produeix donant lloc a una execució incorrecte.

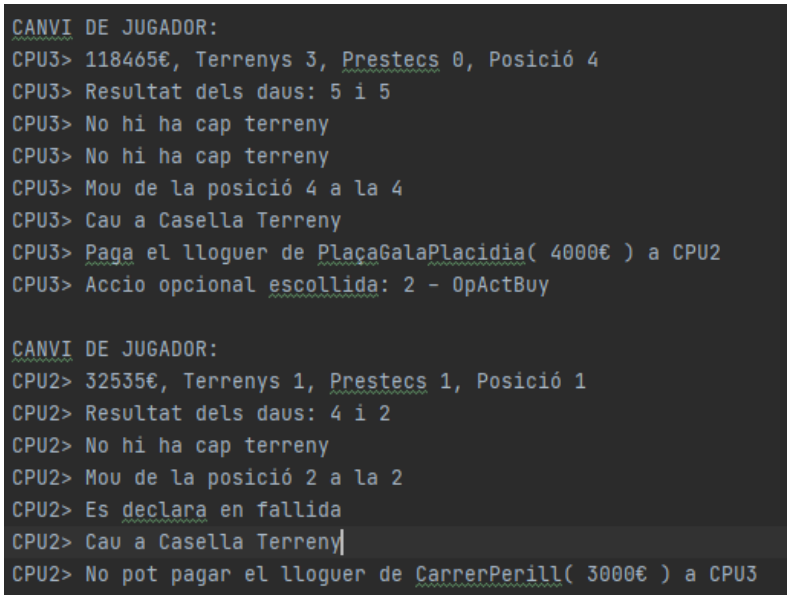
```
if (board.isBankrupt(active_player,field.getRent(), aux: this)) {
    output.fileWrite( line: active_player.getName() + "> No pot pag
    board.transferProperties(active_player,field.getOwner(), mov
    active_player.goToBankruptcy();
}
else {payRent();}
```

Solució: Crida recursiva a la funció en aquest cas

Segon error: Ens trobem amb un altre problema amb la lògica de sortides de la funció de bancarrota

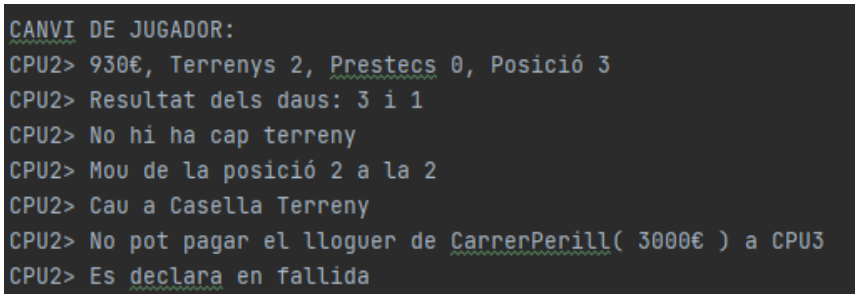
```
public boolean isBankrupt(Player current_player, int pay_amount, Movement aux){
    OutputManager output = aux.getOutput();
    if(!current_player.getLuckCards().isEmpty() || !current_player.getFields().isEmpty()) {
        boolean sell_action_done = false;
        boolean card_action_done = false;
        while (!sell_action_done && !card_action_done) {
            Scanner scan = new Scanner(System.in);
            int missing_money = pay_amount - current_player.getMoney();
            int option_nr = selectOption(current_player,pay_amount, missing_money,scan);
            if (option_nr == 1) {
                while (!sell_action_done) {
                    sell_action_done = tryToSellToAlive(current_player,aux);
                }
            }
            else if (option_nr == 2) {
                card_action_done = tryToRunCardToAlive(current_player,missing_money,scan);
            }
            else {
                System.out.println("El jugador " + current_player.getName() + "s'ha declarat en fallida");
                output.fileWrite( line: current_player.getName() + "> Es declara en fallida");
                current_player.goToBankruptcy();
                return true;
            }
        }
        if(pay_amount > current_player.getMoney()){ return true; }
        else return false;
    }
    return true;
```

Solució: canviar els valors de retorn de la funció



Funcionament ja correcte

Tercer error. El ordre de sortida quan el jugador es declara en fallida està malament ja que primer cau en la casella i després es declara en fallida (veure imatge superior)



Solució: canviar ordre de sortida

TEST4: t4_rules.json t4_board.json | t4_log.txt

En aquesta prova comprovarem el primer dels casos extrems. En aquest cas, el joc de proves nomes conté la casella de sortida.

La execució a priori es correcte si els jugadors tenen una recompensa econòmica. Però, si canviem aquesta recompensa per un terreny, ens trobem amb varies modificacions necessàries.

El primer de aquests, es un bucle infinit de la partida ja que aquesta no acaba mai. Això ho podem solucionar afegint a mes a mes del límit econòmic per acabar la partida, un límit de 500 torns.

```
if (turns > 500) {
    System.out.println("S'ha asolit el maxim de torns posibles i per
    return true;
}
```

Solució a la funció endGame

Afegim també, les sortides de informació en cas de que la partida arribi a aquest estat

```
CPU3> Cau a Casella Sortida
CPU3> Accio opcional escollida: 2 - OpActBuy
CPU3> Cap jugador amb propietats
MAXIM DE TORNS ASOLITS: FINAL PARTIDA
```

```
------ TURNS: 501--------
```

Ens trobem també que en la funció randomField de board, al no tenir cap terreny ens salta un error al general el numero random.

```
Random rand = new Random();
int aux_nr = rand.nextInt( bound: boxes_nr-1)+1;
BoxField aux_field = null;
```

Per solucionar això, generem el numero dins de la comprovació de si hi ha terreny disponibles

```
Random rand = new Random();
BoxField aux_field = null;
if(haveAvailableFields()) {
    int aux_nr = rand.nextInt( bound: boxes_nr-1)+1;
    Box aux_box = board.get(aux_nr);
    boolean have_owner = true;
    if (aux_box.getType().equals("FIELD")) {
        aux_field = (BoxField) aux_box;
        if (!aux_field.isBought()) {
            have_owner = false;
        }
    }
```

TEST5: t5_rules.json t5_board.json | t5_log.txt

En aquest comprovem el segon dels casos extrems. Assignarem a els jugadors, una quantitat de diners inicials de 0€ i cap recompensa econòmica per passar per la casella de sortida.

El primer error amb el que ens trobem es al caure en una casella de sort i que ens surti una carta de tipus multa. Se'ns genera un bucle infinit en la funció de fallida.

```
CPU3> Mou de la posició 18 a la 18
CPU3> Cau a Casella Sort
CPU3> TE MULTAN POR USAR EL MÓVIL MIENTRAS CONDUCES. PAGA 1500€
CPU3> Es declara en fallida
CPU3> Es declara en fallida
CPU3> Es declara en fallida
```

Per solucionar-ho realitzem una crida recursiva a la funció de executar quan el jugador ha aconseguit els diners necessaris i quan no, sortim del programa

Un cop això, fet, el joc de proves funciona be. Els jugadors cauen en bancarrota ràpid degut a la falta de diners. Ja que quan passen per la casella de sortida reben terrenys i fa que després no pugin pagar. Una possibilitat remota de que duri, seria que mes de un jugador guanyes diners gracies a algun terreny i sumat de cartes de Charge i amb això pogues aguantar. Per això, si fem una execució amb 3 jugadors dura 5 tors, però si la fem amb 12 hauria de durar mes.

Executant amb 12 jugadors, ens trobem una altre vegada amb un bucle a la funció de fallida.

```
System.out.println("No tens diners suficients per fer front a la multa");
if (board.isBankrupt(current_player,quantity,aux)) {
    board.transferProperties(current_player, player_get: null,aux);
    end = true;
}
else {
    execute(board,current_player,aux);
    System.out.println("Has aconseguit els diners suficients");
    end = true;
}
```

Solució: error de lògica interna del booleà retornat de bancarrota, i afegir un canvi de variable després de la crida recursiva per eliminar un bucle. Això s'ha de fer a la classe CardFine que es on salta el error.

Un cop fet això i executem amb 12 jugadors, la partida acaba en 7 torns

En trobem però amb algun altre error:

1.- La CPU demana préstecs a un jugador amb 0€ i per tant aquests nomes pot oferir 0€ i en genera un préstec sense sentit

La CPU decideix si necessita demanar un préstec segons la quantitat de diners que te. Llavors, quan ha de seleccionar a el jugador, ja no seleciona a un que no te diners a no ser que cap tingui diners. Podem obviar aquest

problema, ja que no es un error del joc (sempre es pot demanar un préstec de 0€ per poc sentit que tingui), no interfereix amb la execució correcte del joc i la solució implicaria comprovar cada vegada si hi ha un jugador amb diners. Una funció poc funcional i efectiva que nomes es produirà si tots els altres jugadors tenen exactament 0€ i no han caigut en bancarrota (una situació molt improbable) i que afegir-la augmentaria el temps de resposta de la CPU innecessàriament.

```
case "OpActLoan":
    if (player.getMoney() <= 5000)
        bestMoves.put(pos_iterator, player.getMoney() / 1000 * 20);
    else bestMoves.put(pos_iterator, 0);
    break;
```

2.- El joc acaba quan encara el 2, 7 i 10 estàn actius però tot i aixis declara el jugador 10 com a guanyador

```
if (board.isBankrupt(current_player,quantity,aux)) {
    board.transferProperties(current_player, player_get: null,aux);
    current_player.goToBankruptcy();
}
```

Solució: faltava enviar a el jugador a la bancarrota en la funció execute() de CardFine

3.- Error sortida del log al escriure de quina a quina posició es mou el jugador

```
System.out.println("Et mous de la posicio " + current_player.getPosition() + " a la posicio " + position + "\n");
dev_file.fileWrite( line: current_player.getName() +"> Mou de la posició "+current_player.getPosition()+" a la "+position);
board.movePlayer(current_player,position,start_rewards,dev_file);
```

Solució: canviar ordre de sortida i treure per pantalla abans de moure

TEST6: t6_rules.json t6_board.json | t6_log.txt

En aquest joc comprovem que el joc funciona correctament quan entrem una modalitat de joc que no te implementada. Modifiquem la classe principal perquè ens generi també un fitxer de log de la partida ja que si volem executar consecutivament varies proves, ens ajudarà tenir el arxiu encara que en aquest nomes hi hagi la indicació que no es pot jugar.

```
if(!monopoly.getMode().equals("classica")) {
    OutputManager dev_file = new OutputManager();
    dev_file.fileWrite( line: "No es pot jugar amb aquest mode...");
    System.out.println("No es pot jugar amb aquest mode...");
}
else monopoly.play();
```

TEST7: t7_rules.json t7_board.json | t7_log.txt

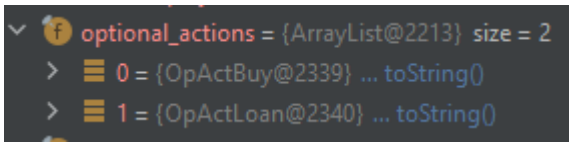
Per últim comprovarem que el joc funciona correctament quan variem les accions no aplicables del fitxer de regles json.

Previament ens donem compta, gracies a les execucions anteriors, que el programa no llegeix be les accions no aplicables i per tant, encara que en els altres testos la acció *COMPRA* estigui bloquejada, el programa deixa al usuari triar-la.

```
//Segon element array "accionsNoAplicables"
JsonArray restricted_actions = j_object.get("accionsNoAplicables").getAsJsonArray();
List<String> restricted_action_list = new ArrayList<>();
for (JsonElement array_element : restricted_actions){
    restricted_action_list.add(array_element.toString());
}
```

Problema amb el toString que s'ha de canviar a getAsString ja que sinó ho guarda com a un array de char i els comparadors següents no funcionen. Això va en el test de diferents accions no aplicables

Comprovem ara que canviant aquets valor en el fitxer json de regles, el programa no les guardi com a possibles accions opcionals. De les 4 possibles accions (Compra, vendre, préstec i sort) eliminem les de vendre i sort. I acabem les dues accions opcionals restants:




```
ABANS DE FINALIZTAR EL TORN, POT FER UNA DE LES SEGUENTS ACCIONS OPCIONALS:
Accions Opcionals:
0 - RES
1 - Comprar: Fer una oferta de compra a un altre jugador
2 - PRÉSTEC: pot demanar una quantitat prestada a un altre jugador
```

CONCLUSIONS

Amb això comprovat podem donar el joc de proves per conclòs. Aquests jocs de proves, avaluen el correcte funcionament de les funcions principals i tracten possibles casos extrems coneguts per el programador. En el millor dels casos, aquests haurien de ser mes extensos i realitzats per varies persones. Però tot i així, simplement amb 7 jocs de probes em pogut solucionar una gran quantitat de errors que sinó hagués sigut extremadament difícil de detectar, o en el cas de que el programa sortís al mercat, provocar un mal funcionament de aquest per el usuari.

Tot i així, la possibilitat de errors, bugs o elements que no hem tingut en compte sempre es possible i com ens ha ensenyat el ús de una gran quantitat de programes i videojocs, es un element comú i molt difícil de erradicar al complet. Els programes creats per us diari del usuari, estan en constant evolució. Tant de correcció de bugs com de aplicació de les funcions. Per això, hem programat el joc de la forma mes organitzada i clara possible, permetent axis que quan ens hem trobat un error en els jocs de proves, hagin sigut molt fàcils de solucionar i nomes hem hagut de tocar una o com a molt dues funcions.