A3.P3: Hunt the Wumpus

Instruccions inicials

Creeu un projecte anomenat ***A3P3WumpusProject***. A dins, creeu-hi els *packages* **inscaparrella.utils**, **inscaparrella.model**, **inscaparrella.controller** i **inscaparrella.view** amb el programa **WumpusMain.java**.

Aquesta pràctica consistirà a implementar el joc "*Hunt the Wumpus*" i l'heu de fer en grups de 3 persones. Al capdamunt del programa, just a sobre de la línia `public class WumpusMain`, indiqueu, amb un comentari multilínia, el nom i els cognoms de tots els membres del grup (Figura 1).

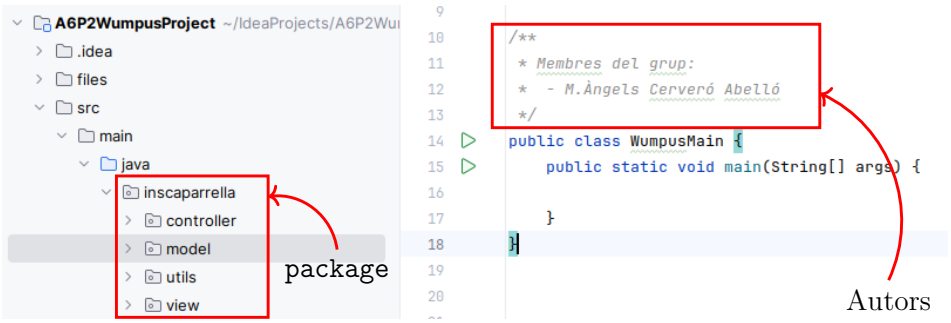


Figura 1: Configuració del *package* i del nom dels membres del grup

Instruccions d'entrega

Per entregar aquesta pràctica, un membre de l'equip ha de comprimir la carpeta del projecte en format **ZIP** (**si m'ho envia en RAR no ho podré obrir!**) i pujar-la al Moodle.

Normes a tenir en compte per a poder fer l'exercici

1. Si la pràctica no complia, no es corregirà i la seva nota serà un 0.
2. Si la pràctica llença alguna excepció, no es corregirà i la seva nota serà un 0.
3. Cal entregar la carpeta de tot el projecte en format ZIP.
4. Si hi ha fitxers duplicats (més d'un fitxer pel mateix exercici o per la mateixa classe) o hi ha dubtes sobre quin fitxer correspon a un exercici/classe, la pràctica no es corregirà i rebrà una nota de 0.
5. No es pot fer ús de cap sistema d'intel·ligència artificial, sigui online o local (ChatGPT, Copilot, Gemini, IA instal·lada localment, etc.).
6. No es copiar el codi de companys ni tampoc deixar-se copiar
7. No es pot utilitzar la comanda `break` per interrompre un control de fluxe iteratiu (`while`, `do-while` o `for`); si una pràctica ho utilitza, encara que només sigui 1 cop, no es corregirà i la seva nota serà un 0 (alerta, sí que es pot utilitzar el `break` per al `switch-case`)
8. En una funció/mètode només es pot posar una única instrucció `return`, la qual ha de ser la única instrucció dins del cos de la funció; si una pràctica té funcions/mètodes amb múltiples `return`, encara que només en sigui 1, no es corregirà i la seva nota serà un 0.
9. Un procediment/mètode no pot tenir cap `return`; si una pràctica té un procediment/mètode amb un `return`, encara que només sigui 1, no es corregirà i la seva nota serà un 0.
10. No es pot utilitzar un `while(true)` ni un `do-while(true)` ni cap altre tipus de condició que faci que les iteracions siguin infinites; si una pràctica té qualsevol d'aquestes condicions, encara que només es faci servir 1 cop, no es corregirà i la seva nota serà un 0.
11. No es poden utilitzar variables globals; si una pràctica en té, no es corregirà i la seva nota serà un 0.
12. Els procediments, les funcions i els mètodes no han de fer cap mena d'interacció amb l'usuari (`System.out` o `Scanner keyboard`) ja que tota la interacció ha de fer-se des del main; si una pràctica fa la interacció amb l'usuari fora del main, encara que només sigui 1 cop, no es corregirà i la seva nota serà un 0.

Diagrama de classes UML

Figura 2: Diagrama de classes UML de la pràctica

Programa WumpusMain

Aquest programa s'ha d'implementar utilitzant la programació orientada a objectes (POO) i seguint el diagrama UML que mostra la Figura 2. Les funcionalitats bàsiques del programa són

1. carregar una partida ja existent i jugar-hi;
2. crear una nova partida, que es guardarà a fitxer, i jugar-hi

Així doncs, el programa s'iniciarà demanant a l'usuari la tasca que vol executar, presentant per pantalla el següent menú una i altra vegada:

```
~~~ HUNT THE WUMPUS ~~~
    0. Sortir
    1. Carregar partida
    2. Crear nova partida
```

Figura 3: Menú inicial

Pel que fa als fitxers que defineixen els laberints de les partides "*Hunt the Wumpus*", tots estaran guardats dins dels directori `files`, el qual contindrà els taulers de partides noves (vegeu la Figura 4 com a exemple de la codificació d'aquests fitxers). La nomenclatura d'aquests documents seguirà la norma "`id_wumpus.txt`".

Aquest directori s'adjunta en aquest enunciat i cal que el situeu dins del vostre projecte. A més a més, a dins hi trobareu el fitxer `wumpus1.txt`.

Obre ▾ ⊞ wumpus1.txt
~/IdeaProjects/A6P2WumpusProject/files

```
1  N N N N N N N N N N N N P
2  N N N N N N N N N N N N N
3  N P N N N N N N N N N N N
4  N N N N N N N N N N N N N
5  N N W N N N N N N N N N N
6  N N P N N N N N P N N N N
7  N N N N N N N N N N N N N
8  N N N N P P N N N P N N N
9  N N N N N N N N N N N N N
10 N N N N P N N N N N N N N
11 N N N N N N N N N N N N N
12 N W N N N N N N N N N N N
13 2 4
14 2 3 2 7 2 10 3 2 4 6 5 5 6 11 7 12 8 6 9 11 10 1 10 8 11 2 11 8
```

Figura 4: Partida nova `wumpus1.txt`

Opció 1: carregar una nova partida

Si el jugador escull l'opció de jugar una partida nova, caldrà demanar-li el nom del fitxer al qual vol jugar. Un cop obtinguda aquesta informació, es carregarà el laberint i s'iniciarà la partida, de tal manera que, a cada torn, es mostrarà el laberint en el seu estat actual i el menú contextual que mostra la Figura 5b.

Opció 2: crear una nova partida

Si el jugador escull l'opció de crear i jugar una partida nova, caldrà

1. demanar-li el nom del fitxer on voldrà guardar les dades del nou laberint;
2. crear el nou laberint;
3. guardar-lo al fitxer indicat i
4. iniciar la partida.

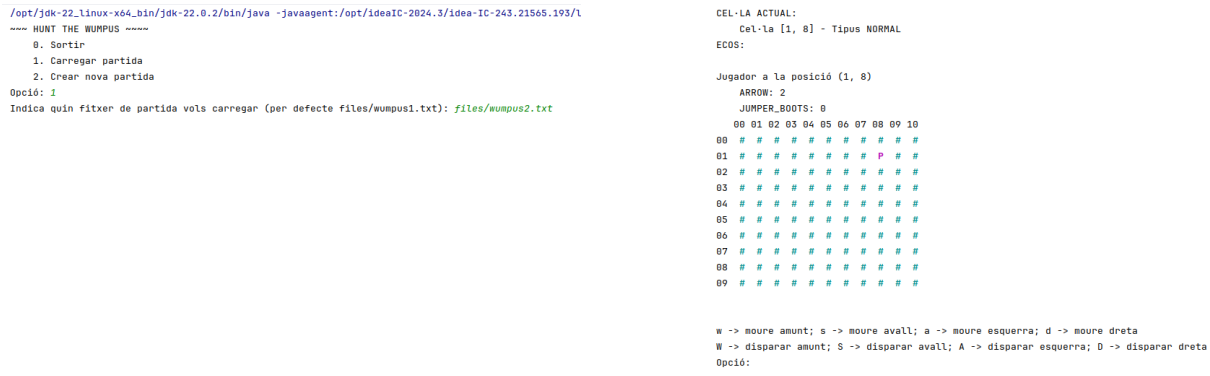
A cada torn, es mostrarà el laberint en el seu estat actual i el menú contextual que mostra la Figura 5b.

Flux de joc, tant per l'opció 1 com per l'opció 2

Mentre el joc no acabi, s'aniran fent tirades i, per cada tirada, s'hauran de processar els següents passos:

1. Mostrar el laberint en l'estat actual, conjuntament amb el menú contextual (vegeu la Figura 5b)
   - si es prem alguna de les tecles la tecla `w`, `s`, `a` o `d`, es moura el jugador a la nova cel·la indicada (`w`-amunt, `s`-avall, `a`-esquerra, `d`-dreta)
   - si es prem alguna de les tecles la tecla `W`, `S`, `A` o `D`, es dispararà una fletxa a la cel·la indicada (`W`-amunt, `S`-avall, `A`-esquerra, `D`-dreta)
2. En cas que el jugador decideixi disparar una fletxa, s'haurà d'informar per pantalla si ha aconseguit caçar al *Wumpus* i guanyar el joc. En cas que no l'hagi pogut caçar, no es mostrarà cap missatge per pantalla però caldrà tenir en compte que, potser, el monstre s'ha espantat i, per tant, ha fet un bot i ha canviat de cel·la
3. En cas que el jugador decideixi moure's, intentarà recórrer la cel·la:
   (a) si no la pot travessar (l'ha enxampat el *Wumpus* o ha caigut en un pou), el joc acabarà amb *GameOver*;
   (b) si hi troba un ratpenat es veurà transportat a una nova cel·la i haurà d'anar repetint el punt 3 fins que el caigui a una cel·la que no tingui cap ratpenat;
   (c) si la cel·la es pot recórrer i no conté cap ratpenat, tornar al pas 1

El joc acabarà quan el jugador hagi pogut caçar al *Wumpus* o quan el jugador acabi malferit, sigui perquè l'ha atacat el *Wumpus* o perquè ha caigut per un pou. Quan això passi, es tornarà al menú inicial (vegeu el Figura 3).



(a) Procés per iniciar una nova partida

(b) Menú contextual per gestionar el fluxe de joc

Figura 5: Exemplificació de l'inici d'una partida nova

Jerarquia `Cell`

La jerarquia `Cell` representa tots els tipus de cel·les que es poden trobar dins del laberint del *Wumpus*:

- `NormalCell`
- `PowerUpCell`
- `WellCell`

Superclasse `Cell`

Creeu la classe `Cell` dins del *package* **inscaparrella.model**

Atributs
De cada cel·la se'n vol guardar les dades següents:

- Posició del laberint on es troba (fila i columna): `row` i `col`
- Tipus de cel·la (aquest atribut ha de ser de tipus `CellType`): ctype
- Si la cel·la ja ha estat oberta (l'usuari hi ha passat algun cop): `open`

Els tipus de cel·la poden ser

- `CellType.NONE`
- `CellType.NORMAL`
- `CellType.POWERUP`
- `CellType.WELL`

Mètodes

- **Constructor per defecte**
  Inicialitza una cel·la tancada de tipus `CellType.NONE` a la posició `(-1, -1)`.
- **Constructor parametritzat**
  Rep per paràmetre la posició de la cel·la i hi crea una cel·la tancada de tipus `CellType.NONE`.
- **Constructor còpia**
  Rep per paràmetre la cel·la que ha de copiar.
- **Getters dels paràmetres `ctype` i `open`**
- `void openCell()`
  Funció que obrirà la cel·la (posarà el paràmetre `open` a `true`)
- `String emitEcho()`
  Funció abstracta (s'haurà d'implementar a les subclasses).
- `boolean isDangerous()`
  Funció abstracta (s'haurà d'implementar a les subclasses).

- `String toString()`
  Retornarà la representació d'aquesta cel·la en format `String`. De cada cel·la se'n vol conèixer la posició, tal com mostra l'exemple:

```
Cel·la [0, 1]
```

- `boolean equals(Object obj)`
  Retornarà `true` si aquesta (`this`) cel·la és igual a la passada per paràmetre (`obj`) i `false` en cas contrari.

Subclasse `NormalCell`

Creeu la classe `NormalCell` dins del *package* **inscaparrella.model**, la qual representa una cel·la normal (`CellType.NORMAL`).

Atributs
Per cada cel·la normal cal saber si està habitada pel *Wumpus*, per un ratpenat o, per contra, està buida, per tant, tindrà un nou atribut:

- Tipus d'habitant (aquest atribut ha de ser de tipus `InhabitantType`): itype

Els tipus d'habitants poden ser

- `InhabitantType.NONE`
- `InhabitantType.WUMPUS`
- `InhabitantType.BAT`

Mètodes

- **Constructor per defecte**
  Crida al constructor més adient de la superclasse per crear una cel·la de tipus `CellType.NORMAL` sense cap habitant `InhabitantType.NONE`.
- **Constructor parametritzat**
  Rep per paràmetre la posició de la cel·la.
  Crida al constructor més adient de la superclasse per crear una cel·la de tipus `CellType.NORMAL` a la posició indicada per paràmetre i sense cap habitant (`InhabitantType.NONE`).
- **Constructor còpia**
  Rep per paràmetre la cel·la que ha de copiar.
  Crida al constructor més adient de la superclasse i crea una nova cel·la exactament igual a la del paràmetre.
- **Getter i Setter del paràmetre** `itype`
- `String emitEcho()`
  - Si la cel·la no està habitada, retornarà l'`String` buit;
  - si està habitada pel *Wumpus*, retornarà el so `"gggrrr... gggGGGGGRRRRRrrr..."` i
  - si està habitada per un ratpenat, retornarà el so `"Flap, flap, flap"`.
- `boolean isDangerous()`
  Retornarà `true` en cas que la cel·la estigui habitada i `false` en cas contrari.

- `String toString()`
  Sobrecarregarà el mètode de la superclasse per tal d'afegir el tipus de cel·la i l'habitant que conté:
  - si és una cel·la buida, crearà la representació
    `Cel·la [0, 1] - Tipus NORMAL`;
  - si és una cel·la habitada amb el *Wumpus*, crearà la representació
    `Cel·la [0, 1] - NORMAL (habitada pel Wumpus)` i
  - si és una cel·la que té un ratpenat, crearà la representació
    `Cel·la [0, 1] - Tipus NORMAL (habitada per un ratpenat)`.
- `boolean equals()`
  Sobrecarregarà el mètode de la superclasse per tal de poder afegir la comparació de l'atribut `itype` a la comprovació d'igualtat entre cel·les.

Subclasse `PowerUpCell`

Creeu la classe `PowerUpCell` dins del *package* **inscaparrella.model**, la qual representa una cel·la que dóna poder a l'usuari (`CellType.POWERUP`)).

Atributs
De cada cel·la de tipus poder se'n vol saber quin *PowerUp* dóna a l'usuari: `power`. Aquest *PowerUp* pot ser qualsevol dels següents:

- `PowerUp.NONE`
- `PowerUp.ARROW`
- `PowerUp.JUMPER_BOOTS`

Mètodes

- **Constructor per defecte**
  Crida al constructor més adient de la superclasse per crear una cel·la buida de tipus `CellType.POWERUP` i amb un poder inicialitzat de manera aleatòria.
- **Constructor parametritzat**
  Rep per paràmetre la posició de la cel·la.
  Crida al constructor més adient de la superclasse per crear una cel·la buida, de tipus `CellType.POWERUP`, amb un poder inicialitzat de manera aleatòria i a la posició indicada per paràmetre.
- **Constructor còpia**
  Rep per paràmetre la cel·la que ha de copiar.
  Crida al constructor més adient de la superclasse i crea una nova cel·la exactament igual a la del paràmetre.
- `void createPowerUp()`
  Aquest mètode és privat i inicialitza l'atribut `power` de manera aleatòria (aquesta inicialització no pot tonar a donar un *PowerUp* de tipus `NONE`).
- `PowerUp consumePowerUp()`
  Si la cel·la està oberta, retorna el `power` que conté i el torna a posar a `PowerUp.NONE`; si la cel·la està tancada, retorna `PowerUp.NONE`.
- `String emitEcho()`
  Si la cel·la no té cap `power`, retornarà l'`String` buit i si té algun `power`, retornarà el so `"Clic, clic..."`.

- `boolean isDangerous()`
  Sempre retornarà `false`.
- `String toString()`
  Sobrecarregarà el mètode de la superclasse per tal d'afegir el tipus de cel·la i el poder que conté:
  - si és una cel·la buida, crearà la representació
    `Cel·la [0, 1] - Tipus POWERUP`;
  - si és una cel·la que conté una fletxa, crearà la representació
    `Cel·la [0, 1] - Tipus POWERUP (concedeix el poder ARROW)` i
  - si és una cel·la que té unes botes, crearà la representació
    `Cel·la [0, 1] - Tipus POWERUP (concedeix el poder JUMPER BOOTS)`.
- `boolean equals()`
  Sobrecarregarà el mètode de la superclasse per tal de poder afegir la comparació de l'atribut `power` a la comprovació d'igualtat entre cel·les.

Subclasse `WellCell`

Creeu la classe `WellCell` dins del *package* **inscaparrella.model**, la qual representa una cel·la que conté un pou.

Atributs
Una cel·la de tipus pou no necessita emmagatzemar cap dada addicional.

Mètodes

- **Constructor per defecte**
  Crida al constructor més adient de la superclasse per crear una cel·la buida de tipus `CellType.WELL`.
- **Constructor parametritzat**
  Rep per paràmetre la posició de la cel·la.
  Crida al constructor més adient de la superclasse per crear una cel·la buida de tipus `CellType.WELL` a la posició indicada per paràmetre.
- **Constructor còpia**
  Rep per paràmetre la cel·la que ha de copiar.
  Crida al constructor més adient de la superclasse i crea una nova cel·la exactament igual a la del paràmetre.
- `String emitEcho()`
  Retornarà el so `"FFFFfffff..."`.
- `boolean isDangerous()`
  Sempre retornarà `true`.
- `String toString()`
  Sobrecarregarà el mètode de la superclasse per tal d'afegir el tipus de cel·la
  `Cel·la [0, 1] - Tipus WELL`;

Classe `WumpusLaberynth`

La classe `WumpusLaberynth` representa el laberint on viu el *Wumpus* i l'heu de crear dins del *package* `inscaparrella.model`

Atributs
Del laberint en volem guardar la següent informació:

- El laberint en si mateix (aquest atribut ha de ser de tipus `ArrayList<ArrayList<Cell>>`): `laberynth`
- La posició de la cel·la on es troba el jugador actualment (un array de 2 posicions): `ppos`
- La posició actual del *Wumpus* (un array de 2 posicions): `wumpuspos`
- La posició actual dels ratpenats (un array): `batspos`

Mètodes

- **Constructor per defecte**
  Inicialitza el laberint buit i posa els arrays `ppos`, `wumpus` i `batspos` a `null`.
- **Getter i Setter del paràmetre** `laberynth`
  Ambdòs mètodes han de fer *deep copy*. El *setter*, a més a més, també s'encarregarà d'inicialitzar els atributs `wumpuspos` i `batspos` segons les dades del laberint que li hàgin passat per paràmetre.
- `void createNewLaberynth`
  Aquest mètode crearà un laberint completament nou i aleatori. Per fer-ho cal tenir en compte el següent:
  - pot tenir qualsevol forma (quadrada, vertical i horitzontal), però la seva mida mínima serà de `5x5` i la màxima de `15x15`;
  - com a mínim tindrà 2 cel·les de tipus `CellType.WELL` i, com a màxim, en tindrà un 5% del total de cel·les;
  - com a mínim tindrà 2 cel·les de tipus `CellType.POWERUP` i, com a màxim, en tindrà un 10% del total de cel·les i
  - com a mínim tindrà 2 ratpenats i, com a màxim, en tindrà un 10% del total de cel·les de tipus `CellType.NORMAL`

  Un cop finalitzada la creació, els atributs `laberynth`, `wumpuspos` i `batspos` han d'estar inicialitzats correctament i, en canvi, `ppos` s'ha de mantenir a valor `null`.
- `int[] getInitialCell()`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit), inicialitza `ppos` a la posició de la cel·la inicial, obre la cel·la i en retorna la posició. Aquesta cel·la es calcula de manera aleatòria, de tal manera que ha de ser de tipus `CellType.NORMAL` i no ha de tenir cap habitant.
  Si `laberynth` no ha estat inicialitzat retorna `null`.
- `int[] movePlayer(MovementDirection dir)`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit), `ppos` no és `null` i el moviment no surt dels límits del laberint, mou la posició del jugador `ppos` una posició cap a la direcció indicada pel parametre `dir`, obre la nova cel·la i en retorna la posició. En cas contrari, no mou la posició i retorna `null`.
  El paràmetre `dir` pot rebre els valors següents:
  - `MovementDirection.UP`: cal moure el jugador una fila cap a dalt;
  - `MovementDirection.DOWN`: cal moure el jugador una fila cap a baix;
  - `MovementDirection.LEFT`: cal moure el jugador una columna cap a l'esquerra i
  - `MovementDirection.RIGHT`: cal moure el jugador una columna cap a la dreta
- `Danger getDanger()`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit) i `ppos` no és `null`, retorna el perill que enfronta el jugador des de la cel·la `ppos`. El valor de retorn pot ser:
  - `Danger.NONE`: si la cel·la `ppos` on es troba el jugador no és perillosa;

  - `Danger.WELL`: si la cel·la `ppos` és de tipus `CellType.WELL`;
  - `Danger.WUMPUS`: si la cel·la `ppos` conté el *Wumpus* i
  - `Danger.BAT`: si la cel·la `ppos` conté un ratpenat.

  En cas contrari, retorna `Danger.NONE`.

- `PowerUp getPowerUp()`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit), `ppos` no és `null` i, a més a més, la cel·la corresponent és de tipus `CellType.POWERUP` retorna el poder que conté. En cas contrari, retorna `PowerUp.NONE`.

- `int[] batKidnapping())`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit), `ppos` no és `null` i la cel·la corresponent conté un ratpenat, mou el jugador a una nova cel·la aleatòria (no pot ser la mateixa on es troba actualment) i en retorna la posició. En cas contrari, no fa res i retorna `null`.

- `boolean shootArrow(ShootDirection dir)`
  Si `ppos` no és `null`, retorna `true` en cas que des d'aquesta posició es pugui disparar una fletxa cap a la direcció `dir`, sense sortir del laberint i ferint al *Wumpus*. En cas contrari, retorna `false`.
  Cal tenir en compte que la capacitat de vol de la fletxa no és massa bona i només pot arribar a la casella veïna (només es pot moure la distància d'una cel·la).
  El paràmetre `dir` pot rebre els valors següents:

  - `ShootDirection.UP`: la fletxa es dispara cap a dalt;
  - `ShootDirection.DOWN`: la fletxa es dispara cap a baix;
  - `ShootDirection.LEFT`: la fletxa es dispara cap a l'esquerra i
  - `ShootDirection.RIGHT`: la fletxa es dispara cap a la dreta

- `boolean startleWumpus()`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit) i `ppos` no és `null`, aquest mètode pot espantar al *Wumpus*, de tal manera que si ho aconsegueix, el monstre fa un bot i canvia a una nova cel·la aleatòria. Per fer-ho, segueix els passos següents:

  1. Calcula un valor aleatori, de tal manera que, si és parell el *Wumpus* s'ha espantat i, si és imparell, el *Wumpus* no s'ha espantat i el mètode no fa res i retorna `false`.
  2. Si el *Wumpus* s'ha espantat, calcula la posició de la nova cel·la on cau després de fer el bot, de tal manera que, aquesta nova cel·la
     - ha de ser de tipus `CellType.NORMAL`;
     - ha d'estar buida (no ha de tenir cap habitant) i
     - no pot ser la cel·la on es troba el jugador (`ppos`)

     Finalment, retorna `true`.

  En cas que `laberynth` no estigui carregat o no s'hagi espantat al *Wumpus*, retorna `false`.

- `void moveBats()`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit) i `ppos` no és `null`, aquest mètode mou cadascun dels ratpenats del laberint a una nova cel·la aleatòria, de tal manera que aquesta nova cel·la:

  - ha de ser de tipus `CellType.NORMAL`;
  - ha d'estar buida (no ha de tenir cap habitant) i
  - no pot ser la cel·la on es troba el jugador (`ppos`)

- `String emitEchoes()`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit) i `ppos` no és `null`, retorna un `String` amb tots els ecos que se senten des de la cel·la `ppos`. Des d'una cel·la només es poden sentir els ecos de les seves 8 cel·les veïnes.
  En cas contrari, retorna un `String` buit.

- `String currentCellMessage()`
  Si `laberynth` ha estat carregat/inicialitzat (no està buit) i `ppos` no és `null`, retorna la representació en format `String` de la cel·la `ppos`.
  En cas contrari, retorna un `String` buit.

- `String toString()`
  Representa el laberint en format `String`.
  Detalls que cal tenir en compte:

  - Les cel·les obertes es representaran amb el caràcter `P` si és on es troba el jugador, amb el caràcter `O` si són de tipus `CellType.WELL` i amb un espai en blanc en cas contrari
  - Les cel·les tancades es representaran amb el caràcter `#`

  Nota addicional: per poder comprovar que el joc funciona bé, representeu el `Wumpus` amb el caràcter `W` i els ratpenats amb el caràcter `*`, tant si les cel·les corresponents estan obertes com si estan tancades. Un cop comprovat el bon funcionament, deixeu aquest codi comentat per tal que els professors el puguem utilitzar durant la correcció.

- `boolean checkCorrectCell(int row, int col)`
  Aquest mètode és privat.
  Si la posició `(row, col)` és vàlida dins del laberint, retorna `true`; si no, retorna `false`.

Classe `Player`

La classe `Player` representa el jugador mentre es mou per dins del laberint.
Creeu la classe `Player` dins del *package* `inscaparrella.model`.

Atributs
De cada jugador se'n vol guardar les dades següents:

- La posició del laberint on es troba: `row` i `col`
- Els poders que té acumulats: `powers[]` Aquest array serà de 2 posicions i cada posició representarà el nombre d'unitats de cada poder (*PowerUp*):
  - Posició 0: unitats del poder `ARROW`
  - Posició 1: unitats del poder `JUMPER_BOOTS`

Mètodes

1. **Constructor per defecte**
   Inicialitza un jugador a la posició (-1, -1) amb 2 fletxes (`PowerUp.ARROW`).
2. **Constructor parametritzat**
   Inicialitza el jugador a la posició que rep per paràmetre (casella d'inici) amb 2 fletxes (`PowerUp.ARROW`).

3. `void setStartingCell(int row, int col)`
   Si la posició actual del jugador és la (-1, -1) el situarà a la cel·la d'inici del laberint `(row, col)`. Si no, no farà res.

4. `void move(int row, int col)`
   Mourà al jugador a la posició indicada per paràmetre.

5. `int getPowerUpQuantity(PowerUp power)`
   Retornarà el nombre d'unitats del `PowerUp power` que té el jugador.

6. `boolean usePower(PowerUp power)`
   Si el jugador té unitats del `PowerUp power`, n'utilitzarà 1 (en restarà una) i retornarà `true`. Si no, no farà res i retornarà `false`.

7. `void addPower(PowerUp power)` Incrementarà en una unitat la quatitat del `PowerUp power` que tingui el jugador.

8. `toString()`
   Representarà l'estat del jugador: posició on es troba i quantes unitats de cada poder té.

Classe `LaberynthController`

La classe `LaberynthController` actua com a *controller* de l'aplicació i, per tant, conté tots els atributs i els mètodes necessaris per poder implementar les funcionalitats del joc *Hunt the Wumpus*. Creeu la classe dins del *package* `inscaparrella.controller`.

Atributs
El *controller* necessita emmagatzemar les dades següents:

- El laberint: `laberynth`
- El jugador: `player`
- El missatge ocasionat per l'última cel·la visitada: `traverseMessage`
- L'eco que s'escolta des de l'última cel·la visitada: `echoes`
- Si el joc a acabat: `gameEnded`
- Si el joc acabat s'ha guanyat o perdut: `won`

Mètodes

- `Constructor per defecte`
  Inicialitza un `laberynth` i un `player` buits, els missatges `traverseMessage` i `echoes` a buits i indica que `gameEnded` és `true` i `won` és `false`.
- `boolean loadLabyrinth(String filename)`
  Llegirà el fitxer `filename` i carregarà les dades del laberint a memòria segons el format que mostra la Figura 6.
- `boolean startGame()`
  Demana a `laberynth` la cel·la d'inici i si és `null`, retorna `false`.
  En cas contrari,

  1. posa `gameEnded` a `false`,

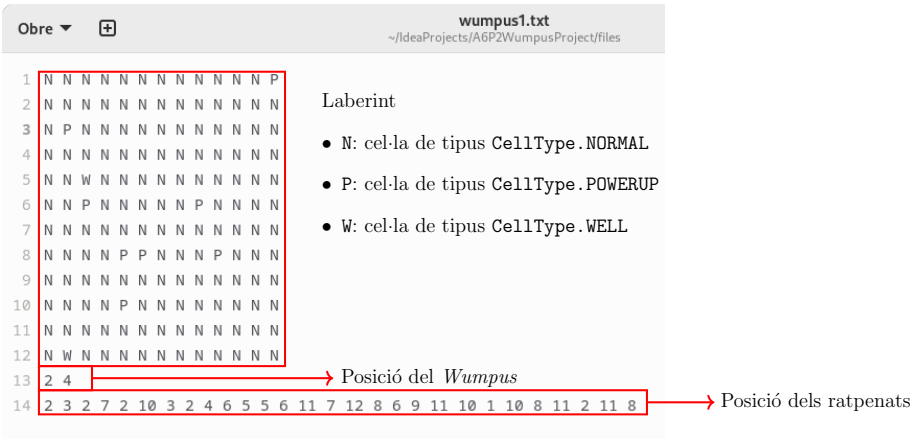


Figura 6: Format del fitxer de partida

2. posa `won` a `false`,
3. posiciona el `player` a la cel·la d'inici,
4. inicialitza `traverseMessage` amb el missatge corresponent a la cel·la inicial,
5. inicialitza `echoes` segons els ecos que se senten des de la cel·la inicial i
6. retorna `true`.

- `void movePlayer(MovementDirection dir)`
  Si `gameEnded` és `true`, no farà res.
  Si no, intentarà moure el jugador segons el moviment indicat. Si el `player` s'ha pogut moure li aplicarà l'efecte de la nova cel·la i si, un cop aplicat aquest efecte, `gameEnded` encara és `false`, tots els ratpenats de `laberynth` canviaran de posició i s'actualitzaran els `echoes`.

- `void huntTheWumpus(ShootDirection dir)`
  Si `gameEnded` és `true` o el jugador no té fletxes, no farà res.
  Si no, dispararà una fletxa des de la posició del jugador, reduint-la del seu inventari i seguint la direcció indicada per paràmetre. Si s'aconsegueix ferir al *Wumpus*, `won` i `gameEnded` passaran a ser `true`. En cas contrari, `won` i `gameEnded` es mantindran a `false` i es gestionarà el possible ensurt del *Wumpus*.

- `String getLastTraverseMessage()`
  Retorna el valor de l'atribut `traverseMessage`.

- `String getLastEchoes()`
  Retorna el valor de l'atribut `echoes`.

- `boolean isGameEnded()`
  Retorna `true` si el joc s'ha acabat i `false` en cas contrari.

- `boolean isGameWon()`
  Retorna `true` si el joc s'ha guanyat i `false` en cas contrari.

- `String toString()`
  Representa l'estat del joc en format `String`.
  Cal tenir en compte que, sobre el dibuix del laberint, s'hi afegirà la representació de l'estat del jugador.

- `void traverseCell()`
  Aquest mètode és privat.
  Aplica l'efecte de la cel·la on es troba el `player`, per tant:

1. en cas que la cel·la sigui de tipus `CellType.NORMAL` i contingu el *Wumpus* o sigui de tipus `CellType.WELL` i no es pugui saltar, `gameEnded` passarà a `true`;
2. en cas que la cel·la sigui de tipus `CellType.NORMAL` i contingu un ratpenat, el ratpenat mourà el `player` a una nova cel·la aleatòria i, per tant, caldrà tornar a cridar el mètode `traverseCell()` una i altra vegada fins que `player` caigui a una cel·la que no tingui cap ratpenat;
3. en cas que la cel·la sigui de tipus `CellType.POWERUP` s'atorgarà el poder que contingui al jugador.

A més a més, inicialitzarà l'atribut `messageTraverse` per tal que mostri el missatge de la cel·la on es troba el jugador (mètode `toString()` de la jerarquia `Cell`) conjuntament amb el següent text:

- `El Wumpus ha atacat i malferit al jugador!`: si el jugador ha anat a parar a la cel·la del *Wumpus*;
- `El jugador ha estat a punt de caure en un pou, però, per sort, portava les JUMPER_BOOTS`: si el jugador ha anat a parar a un pou però l'ha pogut saltar;
- `El jugador ha caigut en un pou i ha quedat malferit!`: si el jugador ha anat a parar a un pou i no l'ha pogut saltar;
- `Un ratpenat s'enduu el jugador!`: si el jugador ha anat a parar a una cel·la amb un ratpenat
- `El jugador ha trobat una unitat del poder XXX`: si el jugador ha trobat algun *PowerUp*. En aquest missatge cal substituir `XXX` per `ARROW` o `JUMPER_BOOTS` segons correspongui

Puntuació

Exercici 1. *(2 punts)*
Implementació de la jerarquia `Cell`

Exercici 2. *(1 punt)*
Implementació de la classe `Player`

Exercici 3. *(3.5 punts)*
Implementació de la classe `WumpusLaberynth`

Exercici 4. *(2.5 punts)*
Implementació del controlador `WumpusController`

Exercici 5. *(1 punt)*
Implementació del programa `WumpusMain`

Nota important: si el joc no funciona, la nota màxima que es pot assolir és un 4

Punts a tenir en compte

Aspectes generals

1. En aquesta pràctica heu de ser rigorosos ja que es tindrà en compte
   - l'ús de constants;
   - la visibilitat d'atributs i mètodes;
   - l'ús de les paraules reservades `this` i `super`;
   - l'ús dels decoradors del codi;
   - la crida correcta dels mètodes i atributs `static`;
   - l'ús de la tècnica *deep copy*

A més a més, també cal que tingueu en compte que **no podeu afegir cap mètode `public`, sinó que només podeu d'utilitzar els que hi ha definits a l'UML**. En canvi, sí que podeu definir els mètodes `private` addicionals que cregueu convenients.

Fitxers adjunts

1. Vídeo d'exemple
2. Els `enum` `CellType`, `Danger`, `InhabitantType`, `PowerUp`, `MovementDirection` i `ShootDirection` i la classe `ConsoleColors`
3. Un laberint d'exemple
4. El diagrama UML