Mad Max Revisited

Albert Oliveras

26 d'abril de 2023

1 Regles del joc

Aquest és un joc per a quatre jugadors. Cada jugador dirigeix un grup de cotxes en les perilloses carreteres de Wasteland. L'objectiu és aconseguir la màxima puntuació possible recopilant bonificacions d'aigua i matant els altres jugadors. No obstant, també cal recollir bonificacions de benzina per poder continuar movent el cotxe i cal evitar xocar amb els pneumàtics.

El joc es juga en un univers cilíndric bidimensional. Després de desplegar-se, aquest univers es pot veure com el resultat de posar el mateix rectangle de manera repetida enganxant-se per la vora vertical:

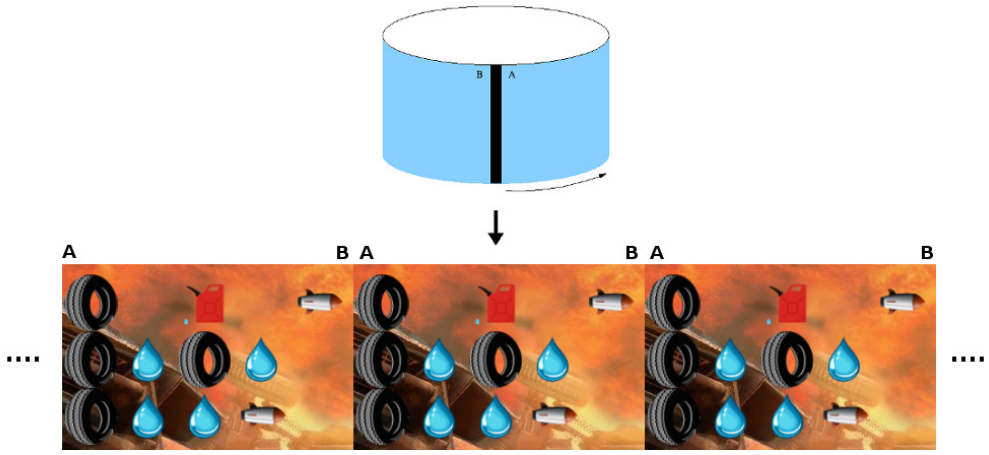

Figura 1: Univers cilíndric després de desplegar-se.

Aquest rectangle repetit és una graella de tamany $n \times m$ de cel·les, on n és el nombre de files i m el nombre de columnes. La cel·la de dalt a l'esquerra és la $(0, 0)$ i la de baix a la dreta la $(n-1, m-1)$. Qualsevol parell (i, j) amb $0 \leq i < n$ determina la posició d'una cel·la de l'univers. Noteu que dues posicions (i, j) i (i, j') tals que $(j - j') \mod m = 0$ es refereixen a la mateixa cel·la. En la resta del document parlarem de i com la *fila* i de j com la *columna* de la posició (i, j), respectivament.

Cada cel·la de l'univers pot estar buida o contenir:

- un pneumàtic, o bé
- un cotxe (dirigit per un dels jugadors), o bé
- un míssil (disparat prèviament per un dels cotxes), o bé
- un bonus d'aigua, o bé
- un bonus de benzina, o bé

- un bonus de míssil.

Els jugadors poden mirar el contingut de qualsevol cel·la de l'univers durant la partida.

Cada jugador dirigeix un nombre de cotxes que és un paràmetre del joc. Cada cotxe té un identificador únic (un nombre). Els cotxes del mateix jugador tenen identificadors consecutius.

1.1 Moure's i disparar

A cada ronda un jugador pot ordenar a qualsevol dels seus cotxes que es moguin en una direcció concreta o que disparin un míssil (però no les dues coses).

En general, el moviment dels elements de l'univers segueix les normes següents:

- Els cotxes es poden moure horitzontalment, verticalment o en diagonal. Més concretament, en un moviment un cotxe pot incrementar la seva fila per -1, 0 or 1, i la seva columna per 0, 1 o 2.
- Per defecte, a cada ronda un cotxe es mou horitzontalment una columna, és a dir, la seva fila es manté i la seva columna s'incrementa en 1.
- Hi ha una *finestra* en la que els cotxes han de romandre dins. Aquesta finestra és un rectangle de dimensions $n \times m_W$, on $m_W \leq m$, i és dinàmica: avança una columna per ronda. Per tant, a la ronda r, el seu cantó superior esquerre es troba en la posició $(0, r)$ i el seu cantó inferior dret està a la posició $(n-1, r+m_W-1)$. A més, un cotxe que es mou en la direcció per defecte roman sempre dins la finestra.

  Si es demana que un cotxe es desplaci a una posició que no sigui a la finestra, la comanda serà ignorada i el cotxe es mourà segons la direcció per defecte.

  En el visualitzador del joc, la finestra és la part de l'univers que es mostrarà. Com que la finestra està presa com a sistema de referència per a la visualització, *aparentment* no es mou; de la mateixa manera, els cotxes que es desplacin per defecte *aparentment* tampoc es mouen, etc.
- Els pneumàtics i les bonificacions no es mouen.
- Els míssils es mouen horitzontalment dues columnes per ronda.
- Els pneumàtics, els bonus i els míssils poden estar fora de la finestra. Noteu que en aquest cas, tot i que no es poden veure, encara existeixen a l'univers.

Un cotxe pot disparar un míssil si en té almenys un en estoc (que després es consumeix). Hi ha un nombre inicial de míssils en l'estoc de cada cotxe.

Aquest estoc es pot ampliar mitjançant bonificacions de míssils. Quan un cotxe dispara, el nou míssil automàticament avança dues columnes des de la posició del cotxe i, a continuació, immediatament després, el cotxe també es mou amb la direcció per defecte.

1.2 Col·lisions en una cel·la

Quan un cotxe i una bonificació coincideixen a la mateixa cel·la, el cotxe consumeix la bonificació (augmentant el nombre de míssils disponibles si es tracta d'un bonus de míssil, el nombre d'unitats de benzina si es tracta d'una bonificació de benzina, o obtenint més puntuació si es tracta d'una bonificació d'aigua). Una vegada consumida una bonificació, si és possible es regenera en una posició aleatòria fora de la finestra. Aquesta posició aleatòria estava anteriorment buida i a la seva la zona circumdant 5×5 no hi ha ni míssils ni cotxes.

En els altres possibles casos de col·lisió, quan coincideixen dos elements a la mateixa cel·la es destrueixen tots dos. En particular, si un dels elements de la col·lisió és un míssil disparat per un cotxe del jugador A i l'altre element és un cotxe d'un jugador diferent B, el jugador A augmenta la seva puntuació.

Quan es mata un cotxe, es regenera si és possible després d'un nombre de rondes, en una posició aleatòria de la finestra. Aquesta posició aleatòria estava prèviament buida i la seva zona circumdant 5×5 està lliure de míssils, cotxes i pneumàtics. El nombre de míssils disponibles és el mateix que abans de morir i les unitats de benzina es determinen per $initial\_gas()$.

Com a consideració final pel que fa a les col·lisions, quan en una ronda un cotxe (o un míssil) es mou d'una cel·la inicial a una cel·la final, en alguns casos es considera que també passa certes cel·les intermèdies. Més específicament:

1. Quan un cotxe es mou de la cel·la (i, j) a la cel·la $(i + 1, j + 1)$, també passa per les cel·les $(i + 1, j)$ i $(i, j + 1)$. Similarment quan es mou a la cel·la $(i - 1, j + 1)$.

2. Quan un cotxe (o un míssil) es mou de la cel·la (i, j) a la cel·la $(i, j + 2)$, també passa per la cel·la $(i, j + 1)$.

3. Quan un cotxe es mou de la cel·la (i, j) a la cel·la $(i + 1, j + 2)$, també passa per les cel·les $(i, j + 1)$, $(i + 1, j + 1)$, $(i + 1, j)$ i $(i, j + 2)$. Similarment quan es mou a $(i - 1, j + 2)$.

4. En els altres casos no passa per cel·les intermèdies.

L'ordre exacte en què aquestes cel·les intermèdies es visiten es pot mirar al map `dir2all` definit a `Utils.cc`.

1.3 Ordre d'execució de les instruccions

Després de recollir les instruccions de tots els jugadors, les següents accions es prenen:

1. Primer de tot, els míssils ja disparats es mouen segons les seves regles.

2. A continuació es determina un ordre aleatori entre els jugadors, i les instruccions per als seus cotxes s'executen en aquest ordre. Les instruccions del mateix jugador s'executen en l'ordre original. Instruccions no vàlides (per exemple, traslladar-se fora dels límits de la finestra) s'ignoren. Si un cotxe rep més d'una instrucció, només es tindrà en compte la primera. Cotxes que no hagin rebut cap instrucció es mouran per defecte, en ordre creixent d'identificador.

3. Després d'executar totes les instruccions, tots els cotxes que estiguin vius veuran les seves unitats de benzina decrementades en una unitat. Després d'això, qualsevol cotxe amb 0 unitats de benzina es destrueix.

4. Si s'escau, els cotxes morts i les noves bonificacions (en aquest ordre) es regeneren.

1.4 Paràmetres de les partides

Una partida ve determinada pels següents paràmetres. Tots ells s'especifiquen en l'arxiu d'entrada de la partdia, però alguns d'ell sempre tenen el mateix valor.

- *number_players*(): nombre de jugadors de la partdia. Fixat a 4.
- *number_rounds*(): nombre de rondes que es jugaran. Fixat a 300.
- *number_rows*(): nombre de files de l'univers (i de la finestra). Variable en $[15, 20]$.
- *number_universe_columns*(): nombre de columnes de l'univers. Variable en $[60, 100]$.
- *number_window_columns*(): nombre de columnes de la finestra. Variable en $[30, 40]$.
- *number_cars_per_player* (): nombre de cotxes de cada jugador. Fixat a 2.
- *number_cars*(): nombre total de cotxes. Fixat a 8.
- *number_rounds_to_regenerate* (): nombre de rondes que cal esperar per a regenerar un cotxe. Fixat a 30.
- *number_missile_bonuses* (): nombre de bonus de míssil a la partida. Variable en $[5, 30]$.
- *number_water_bonuses*(): nombre de bonus d'aigua a la partida. Variable en $[20, 150]$.
- *number_gas_bonuses*(): nombre de bonus de benzina a la partida. Variable en $[5, 15]$.
- *bonus_missiles* (): nombre de míssils extra que s'obtenen en consumir un bonus de míssil. Fixat a 5.

- *bonus_gas*(): nombre d'unitats de benzina extres que s'obtenen en consumir un bonus de benzina. Fixat a 30.
- *water_points* (): nombre punts extra que s'obtenen en consumir un bonus d'aigua. Fixat a 10.
- *kill_points* (): nombre de punts que s'obtenen en matar un cotxe d'un altre jugador. Fixat a 30.
- *initial_gas* (): nombre d'unitats de benzina que es donen a un cotxe quan es regenera. Fixat a 60.

Tots aquests paràmetres són accessibles pels jugadors durant tota la partida.

2 Com programar un jugador

El primer que heu de fer és descarregar-vos el codi font. Aquest inclou un programa C ++ que executa les partides i un visualitzador HTML per veure-les en un format raonable i animat. A més, us proporcionem un jugador "Null" i un jugador "Demo" per facilitar el començament de la codificació del jugador.

2.1 Executar la primera partida

Aquí us explicarem com executar el joc sota Linux, però hauria de funcionar també sota Windows, Mac, FreeBSD, OpenSolaris, ... Només necessiteu una versió recent `g ++`, el `make` instal·lat al sistema, a més d'un navegador modern com Firefox o Chrome.

1. Obriu una consola i feu `cd` al directori on us heu descarregat el codi font.
2. Si, per exemple, teniu una versió de Linux en 64 bits, executeu:

   ```
   cp AIDummy.o.Linux64 AIDummy.o
   ```

   Amb altres arquitectures, cal escollir els objectes adequats que trobareu al directori.
3. Executeu

   ```
   make all
   ```

   per compilar el joc i tots els jugadors. Tingueu en compte que el `Makefile` identifica com a jugador qualsevol fitxer que coincideixi amb `AI*.cc`
4. Es crea un fitxer executable anomenat `Game`. Aquest executable us permet executar una partida mitjançant una commanda com la següent:

   ```
   ./Game Demo Demo Demo Demo -s 30 < default.cnf > default.res
   ```

   Aquesta comanda comença una partida, amb la llavor aleatòria 30, amb quatre instàncies del jugador Demo, al tauler definit a `default.cnf`. La sortida d'aquesta partida es redirigeix a `default.res`.

5. Per veure una partida, obriu el fitxer visualitzador `viewer.html` amb el navegador i carregueu el fitxer `default.res`.

Utilitzeu
`./Game --help`
per veure la llista de paràmetres que es poden usar. Particularment útil és
`./Game --list`
per veure tots els noms de jugadors reconeguts.

En cas que sigui necessari, recordeu que podeu executar
`make clean`
per esborrar l'executable i els objectes i començar la compilació de nou.

2.2 Afegir el vostre jugador

Per crear un jugador nou copieu `AINull.cc` (un jugador buit que proporcionem com a plantilla) a un fitxer nou `AIElquesigui.cc`. A continuació, editeu el fitxer nou i canvieu la línia

#define PLAYER_NAME Null

a

#define PLAYER_NAME Elquesigui

El nom que trieu pel vostre jugador ha de ser únic, no ofensiu i tenir com a màxim 12 caràcters. Aquest nom es mostrarà al lloc web i durant les partides.

A continuació, podeu començar a implementar el mètode virtual *play*(), heretat de la classe base *Player*. Aquest mètode, que serà cridat a cada ronda, ha de determinar les ordres que s'enviaran a les vostres unitats.

Podeu utilitzar definicions de tipus, variables i mètodes a la vostra classe de jugador, però el punt d'entrada del vostre codi serà sempre el mètode *play*().

Des de la classe del vostre jugador també podeu cridar a funcions per accedir a l'estat del taulell, definit a la classe *Board* a `Board.hh`, i per dirigir els vostres cotxes, mireu la classe *Action* definida a `Action.hh`. Aquestes funcions són disponibles al vostre codi usant herència múltiple via la classe *Player* a `Player.hh` . La documentació de les funcions disponibles es pot trobar als arxius header de cada classe. També podeu examinar el codi del jugador "Demo" a `AIDemo.cc`, com un exemple de com usar aquestes funcions. Finalment, val la pena mirar l'arxiu `Utils.hh` per estructures de dades útils.

Tingueu en compte que no heu d'editar el mètode *factory* () de la classe del vostre jugador, ni l'última línia que afegeix el vostre jugador a la llista de jugadors disponibles.

2.3 Jugar contra el jugador Dummy

Per a provar la vostra estratègia contra el jugador Dummy, proporcionem el seu arxiu objecte. D'aquesta manera, no teniu accés al seu codi font però podreu afegir-lo com a jugador i competir contra ell en local.

Per afegir el jugador Dummy a la llista de jugadors registrats, heu d'editar el `Makefile` i posar la variable `DUMMY_OBJ` al valor apropiat. Recordeu que els arxius objecte contenen instruccions binàries per a una arquitectura concreta, pel que no podem proporcionar un únic arxiu.

Consell de pro: demaneu als vostres amics els seus arxius **objecte** (mai codi font!!!) i afegiu-los al vostre `Makefile`!

2.4 Restriccions en enviar el vostre jugador

Quan creieu que el vostre jugador és prou fort per entrar a la competició, podeu enviar-lo al Jutge. Degut a que s'executarà en un entorn segur per prevenir trampes, algunes restriccions s'apliquen al vostre codi:

- Tot el vostre codi font ha d'estar en un sol fitxer (com AIElquesigui.cc).
- No podeu utilitzar variables globals (en el seu lloc, utilitzeu atributs a la vostra classe).
- Només teniu permès utilitzar biblioteques estàndard com `iostream`, `vector`, `map`, `set`, `queue`, `algoritme`, `cmath`, ... En molts casos, ni tan sols cal incloure la biblioteca corresponent.
- No podeu obrir fitxers ni fer cap altra crida a sistema (threads, forks, ...)
- El vostre temps de CPU i la memòria que utilitzeu seran limitats, mentre que no ho són al vostre entorn local quan executeu `./Game`. El temps límit és d'un segon per l'execució de tota la partida. Si exhauriu el temps límit (o si l'execució del vostre codi aborta), el vostre jugador es congelarà i no admetrà més instruccions.
- El vostre programa no ha d'escriure a **cout** ni llegir de **cin**. Podeu escriure informació de depuració a **cerr**, però recordeu que fer això en el codi que envieu al Jutge pot fer malgastar innecessàriament temps de CPU.
- Qualsevol enviament al Jutge ha de ser un intent honest de jugar. Qualsevol intent de fer trampes de qualsevol manera serà durament penalitzat.

3 Consells

- **NO DONEU O DEMANEU EL VOSTRE CODI A NINGÚ**. Ni tan sols una versió antiga. Ni fins i tot al vostre millor amic. Ni tans sols d'estudiants d'anys anteriors. Utilitzem detectors de plagi per comparar els

vostres programes, també contra enviaments de jocs d'anys anteriors. No obstant, podeu compartir arxius objecte.

Qualsevol plagi implicarà **una nota de 0 en l'assignatura** (no només del Joc) de tots els estudiants involucrats. Es podran també prendre mesures disciplinàries addicionals. Si els estudiants A i B es veuen implicats en un plagi, les mesures s'aplicaran als dos, independentment de qui va crear el codi original. No es farà cap excepció sota cap circumstància.

- Abans de competir amb els companys, concentreu-vos en derrotar al "Dummy".
- Llegiu les capçaleres de les classes que aneu a utilitzar. No cal que mireu les parts privades o la implementació.
- Comenceu amb estratègies simples, fàcils d'implementar i depurar, ja que és exactament el que necessitareu al principi.
- Definiu mètodes auxiliars senzills (però útils) i *assegureu-vos que funcionin correctament*.
- Intenteu mantenir el vostre codi net. Això farà més fàcil canviar-lo i afegir noves estratègies.
- Com sempre, compila i prova el teu codi sovint. És *molt* més fàcil rastrejar un error quan només heu canviat poques línies de codi.
- Utilitzeu **cerr** per produir informació de depuració i afegiu `asserts` per assegurar-vos que el vostre codi fa el que hauria de fer. No oblideu eliminar-los abans de pujar el codi, per no fer-lo innecessàriament lent.
- Quan depureu un jugador, elimineu els **cerr**s que tingueu en el codi d'altres jugadors, per tal de veure només els missatges que desitgeu.
- Podeu utilitzar comandes com el `grep` de Linux per tal de filtrar la sortida produïda per `Game`.
- Activeu l'opció `DEBUG` al Makefile, que us permetrà obtenir traces útil quan el vostre programa aborta. També hi ha una opció `PROFILE` que podeu utilitzar per optimitzar codi.
- Si l'ús de **cerr** no és suficient per depurar el vostre codi, apreneu com utilitzar `valgrind`, `gdb` o qualsevol altra eina de depuració.
- Podeu analitzar els arxius produïts per `Game`, que descriuen com evoluciona el taulell a cada ronda.
- Conserveu una còpia de les versions antigues del vostre jugador. Feu-lo lluitar contra les seves versions anteriors per quantificar les millores.
- Assegureu-vos que el vostre programa sigui prou ràpid. El temps de CPU que es permet utilitzar és bastant curt.

- Intenteu esbrinar les estratègies dels altres jugadors observant diverses partides. D'aquesta manera, podeu intentar reaccionar als seus moviments, o fins i tot imiteu-los o milloreu-los amb el vostre propi codi.
- No espereu fins al darrer minut per enviar el jugador. Quan hi ha molts enviaments al mateix temps, el servidor triga més en executar les partides i podria ser ja massa tard!
- La majoria dels paràmetres del joc no canvien, però si la vostra estratègia s'adapta a elles, tindreu un plus de seguretat en cas que hi hagi algun canvi necessari.
- Podeu enviar noves versions del vostra programa en qualsevol moment.
- Si creeu el vostre taulell pel joc, envieu-nos-el abans que la competició i potser l'inclourem!
- Recordeu: mantingueu el codi senzill, compileu-lo sovint i proveu-lo sovint, o us en penedireu.