Factorio Planner

Pau Jimeno Roman

Juny 2024

1 Elements i mecàniques

Factorio, és un joc d'automatització i ampliació de fàbriques en un món 2D basat en una graella. Els elements que el joc posa a disposició del jugador són molts, però per mantenir la simplicitat del model (a hores d'ara "5/3/2024"), només s'han tingut en compte els següents elements i mecàniques.

1.1 Cinta transportadora

Les cintes transportadores o *conveyor belts* serveixen per transportar elements d'un punt a un altre. Aquestes ocupen 1x1 caselles a la graella del món i poden prendre qualsevol de les 4 direccions cardinals. Transporten objectes de la casella actual a la següent casella apuntada per la cinta. Aquestes poden rebre objectes per qualsevol de les tres caselles adjacents diferents de la casella de sortida (la que apunta la cinta). Al joc les cintes consten de dues pistes cadascuna amb la mateixa capacitat de transport d'objectes, però de moment no s'ha tingut en compte aquesta mecànica, ja que de la manera que funciona afegeix molta complexitat.

Figura 1: *Conveyor belt* a la graella del joc

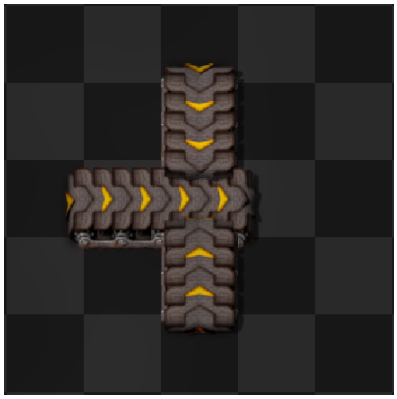

Figura 2: Entrades permeses a una *conveyor belt*

Les cintes també poden interactuar amb altres elements a banda d'elles mateixes, al model (a hores d'ara "5/3/2024") només s'han considerat els *inserters*.

1.2 Inseridor

Els inseridors o *inserters* són braços robòtics que ocupen 1x1 caselles a la graella del món, permeten inserir i treure elements de cintes i *assemblers*. Al joc hi ha més elements amb els quals interactuen, però de moment només s'han considerat aquests dos.
Aquests poden treure elements de cintes que apuntin en qualsevol direcció i inserir objectes a cintes que no apuntin cap a ell, però al model s'ha forçat que un *inserter* no pugui agafar element d'una cinta que apunta cap a ell, ja que això significa que en comptes d'un *inserter* hi pot haver una cinta sent així redundant, així doncs al model, els *inserters* només són útils per crear bifurcacions i inserir o treure objectes d'un *assembler*.

Figura 3: *Inserter* a la graella del joc

Figura 4: Entrades permeses

Figura 5: Sortides permeses

1.3 Assemblador

Els assembladors o *assemblers* ocupen un espai de 3x3 caselles a la graella del món, aquests només poden rebre i treure objectes a través d'*inserters*, tal com anteriorment s'ha explicat.
La funcionalitat dels *assemblers* és convertir els materials d'entrada en objectes refinats, això es fa mitjançant receptes, les quals estan associades a un *assembler* concret i especifiquen quants materials d'entrada (poden ser un o varis), són requerits per produir un material de sortida (sempre n'és un) i també s'indica quant temps es tarda a produir l'objecte.

Figura 6: *Assembler* a la graella del joc

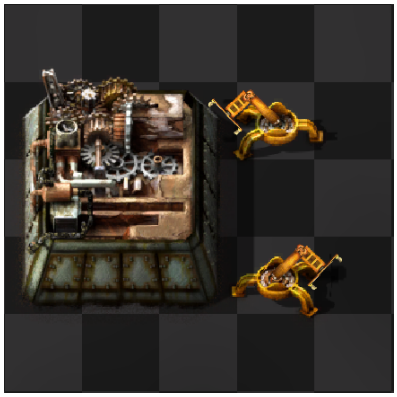

Figura 7: Entrada i sortida d'un *assembler*

Figura 8: Informació de la recepta d'un *assembler*

2 Definició de les restriccions per a la recreació de les mecàniques del joc

2.1 Rutes

Una de les parts més importants del model és definir com la concatenació d'*inserters* i *conveyor belts* conformen una ruta. Perquè una ruta sigui vàlida no pot crear cicles, és a dir que una cinta o *inserter* no pot entrar objectes a una cinta la qual era anterior a ella mateixa. També s'ha de tenir en compte l'orientació dels *inserters* i cintes per complir amb les entrades i sortides vàlides anteriorment descrites. Una ruta també es pot bifurcar i unir. Així doncs, per implementar la noció de ruta s'ha usat la representació incremental on un element que forma part d'una ruta ha de precedir un element amb un valor de ruta inferior a ell i ser precedir per un element amb valor de ruta superior a ell. Per implementar aquesta representació s'ha creat una variable de tipus matriu `route` de mida $width \times height$, el seu domini també és $width \times height$, ja que una ruta pot ocupar com a màxim tota l'àrea del *blueprint*, a banda també cal saber l'orientació dels elements que formen part de la ruta (*inserters* i cintes) les quals es guarden en dues variables una per cada element `conveyor` i `inserter` aquestes de la mateixa manera que la ruta són matrius de mida $width \times height$ i el seu domini `EnumSort('direction', ['empty', 'north', 'east', 'south', 'west'])` on empty significa que no hi ha cap *inserter* o cinta present. Amb aquestes variables ja es poden definir les restriccions que conformen una ruta, principalment en són dues:

2.1.1 Forward increment

Aquesta restricció ens codifica qualsevol element que formi part d'una ruta, ha de tenir una casella adjacent la qual el seu valor de la ruta sigui superior. En aquest cas les caselles adjacents vàlides només són les que estan en la mateixa direcció que la cinta o *inserter* corresponent a la posició de la ruta. A banda també s'ha de tenir en compte que les rutes poden acabar si a la posició de la ruta hi ha un *inserter* i en la direcció on aquest apunta hi ha un *assembler*. A banda una ruta també pot acabar si la posició de la ruta es tracta d'una casella output donada com a entrada del model. En aquests dos casos no cal assegurar que la posició en la direcció del *inserter* el valor de ruta sigui més gran. Finalment, la implementació de la restricció és la següent.

```
    def forward_increment(self):
        forward_increment = []
        for i in range(self.height):
            for j in range(self.width):
                if not self.is_output(i, j):
                    inserter_output = []
                    conveyor_output = []
                    for direction in range(1, self.n_dir):
                        x, y = i + self.dir_shift[direction][0], j + self
                            .dir_shift[direction][1]
                        if 0 <= x < self.height and 0 <= y < self.width:
                            conveyor_output.append(If(self.conveyor[i][j]
                                == self.direction[direction],
                                                      UGT(self.route[x][y
                                                          ], self.route[i
                                                          ][j]), False))

                            inserter_output.append(If(And(self.inserter[i
                                ][j] == self.direction[direction],
                                                          self.assembler[
                                                              x][y] == 0),
                                                      UGT(self.route[x][y
                                                          ], self.route[i
                                                          ][j]), False))

                            inserter_output.append(If(And(self.inserter[i
                                ][j] == self.direction[direction],
                                                          self.assembler[
                                                              x][y] != 0),
                                                     self.route[x][y] ==
                                                          0, False))

                    forward_increment.append(If(UGT(self.route[i][j], 0),
                        Or(conveyor_output+inserter_output), True))
        return forward_increment
```

Cal mencionar que si les rutes només haguessin d'anar d'un punt A a un punt B amb aquesta restricció seria suficient per assegurar que no es creïn cicles i que realment s'està creant una ruta que va del punt A al B, però com que en el nostre cas les rutes s'han de poder bifurcar, unir i arribar des de múltiples punts a múltiples punts, cal la següent restricció.

2.1.2 Backwards decrement

Per poder incloure bifurcacions i unions ens cal una restricció que de manera similar a l'anterior restricció ens asseguri que si una cinta o *inserter* forma part d'una ruta i no es tracta d'un inici de ruta, llavors ha de tenir en una de les caselles veïnes, hi hagi un element de la ruta amb un valor inferior. Aquestes caselles veïnes seran diferents en funció de si l'element de la ruta tracta d'una cinta o un

inserter. Al cas de la cinta aquesta pot rebre input de qualsevol direcció que no sigui la mateixa que ella (figura 2) i en cas del *inserter* aquest només pot rebre input de la casella en la direcció contrària al *inserter* (figura 4), a banda els *inserters* també poder agafar objectes a un *assembler*, així que en aquest cas forçarem que el valor de ruta a la posició del *inserter* sigui 1 (inici de ruta). Així doncs, la restricció queda de la següent manera:

```
    def backward_consistency(self):
        backward_consistency = []
        for i in range(self.height):
            for j in range(self.width):
                if not self.is_input(i, j):
                    inserter_input = []
                    conveyor_input = []
                    for direction in range(1, self.n_dir):
                        x, y = i + self.dir_shift[direction][0], j + self
                            .dir_shift[direction][1]
                        if 0 <= x < self.height and 0 <= y < self.width:
                            conveyor_input.append(If(And(self.conveyor[i
                                ][j] != self.direction[direction],
                                                         self.conveyor[i
                                                             ][j] != self.
                                                             direction[0])
                                                         ,
                                                     And(ULT(self.route[x
                                                         ][y], self.route[
                                                         i][j]),
                                                         UGT(self.route[x
                                                             ][y], 0)),
                                                     False))
                            inserter_input.append(If(And(self.inserter[i
                                ][j] == self.opposite_dir[direction],
                                                         self.assembler[x
                                                             ][y] == 0),
                                                     And(ULT(self.route[x
                                                         ][y], self.route[
                                                         i][j]),
                                                         UGT(self.route[x
                                                             ][y], 0)),
                                                     False))
                            inserter_input.append(If(And(self.inserter[i
                                ][j] == self.opposite_dir[direction],
                                                         self.assembler[x
                                                             ][y] != 0),
                                                     self.route[i][j] ==
                                                         1,
                                                     False))

                    backward_consistency.append(If(UGT(self.route[i][j],
                        0), Or(conveyor_input+inserter_input), True))
        return backward_consistency
```

Amb aquestes dues restriccions assegurem que la ruta no generi bucles i arribi als punts corresponents, però no ens assegura que l'orientació dels elements ni quins elements poden estar interconnectats entre si, així doncs per acabar de definit una ruta cal afegir quines entrades i sortides són vàlides per una cinta i un *inserter*:

2.1.3 Conveyor input i conveyor output

Amb aquestes dues restriccions definim quines entrades i sortides són vàlides per una cinta. Pel que fa a les entrades pot rebre objectes per qualsevol de les posicions adjacents que no estiguin en la mateixa direcció que la mateixa cinta, a més aquestes caselles poden ser tant *inserters* com cintes. A més la cinta o *inserter* que estigui a la casella veïna ha d'apuntar a la cinta, és a dir, la seva direcció pot ser qualsevol menys l'oposada a la cinta. La restricció queda així:

```
    def conveyor_input(self):
        conveyor_input = []
        for i in range(self.height):
            for j in range(self.width):
                if not self.is_input(i, j):
                    direction_clauses = []
                    for direction in range(1, self.n_dir):
                        x, y = i + self.dir_shift[direction][0], j + self
                            .dir_shift[direction][1]
                        if 0 <= x < self.height and 0 <= y < self.width:
                            direction_clauses.append(If(self.conveyor[i][
                                j] != self.direction[direction],
                                                        Or(self.conveyor[
                                                            x][y] == self.
                                                            opposite_dir[
                                                            direction],
                                                            self.inserter[
                                                            x][y] == self.
                                                            opposite_dir[
                                                            direction]),
                                                        False))
                    conveyor_input.append(If(self.conveyor[i][j] != self.
                        direction[0], Or(direction_clauses), True))
        return conveyor_input
```

D'altra una cinta només pot treure elements per la casella adjacent a la direcció a la qual apunta i com bé s'ha mencionat per evitar afegir *inserters* redundants, aquesta casella veïna només pot estar ocupada per una cinta, la direcció de la qual ha de ser qualsevol menys l'oposada a la cinta. La restricció queda de la següent manera:

```
    def conveyor_output(self):
        conveyor_output = []
        for i in range(self.height):
            for j in range(self.width):
                if not self.is_output(i, j):
                    direction_clauses = []
                    for direction in range(1, self.n_dir):
                        x, y = i + self.dir_shift[direction][0], j + self
                            .dir_shift[direction][1]
                        if 0 <= x < self.height and 0 <= y < self.width:
                            direction_clauses.append(If(self.conveyor[i][
                                j] == self.direction[direction],
                                                        And(self.conveyor
                                                            [x][y] != self
                                                            .direction[0],
                                                            self.conveyor
                                                            [x][y] !=
                                                            self.
                                                            opposite_dir
                                                            [direction
```

```
                                                        ]),
                                                    False))
                    conveyor_output.append(If(self.conveyor[i][j] != self
                        .direction[0], Or(direction_clauses), True))

        return conveyor_output
```

2.1.4 Inserter input i Inserter output

Els *inserters* són molt similars a les cintes a l'hora de rebre objectes, però amb dues particularitats, primer només poden rebre objectes de la casella adjacent en la direcció contrària al *inserter* i segons aquesta casella adjacent també pot ser un *assembler*. A l'hora de treure objectes de la mateixa manera que les cintes, només ho pot fer en la casella adjacent que es troba en la mateixa direcció, però en aquesta casella només hi pot haver una cinta que no apunti en la direcció oposada al *inserter* o bé un assembler. Les restriccions són les següents:

```
    def inserter_input(self):
        inserter_input = []

        for i in range(self.height):
            for j in range(self.width):
                if not self.is_input(i, j):
                    direction_clauses = []
                    for direction in range(1, self.n_dir):
                        x, y = i + self.dir_shift[direction][0], j + self
                            .dir_shift[direction][1]
                        if 0 <= x < self.height and 0 <= y < self.width:
                            if not self.is_output(x, y):
                                # The inserter can take input from a
                                    conveyor or an assembler
                                direction_clauses.append(If(self.inserter
                                    [i][j] == self.opposite_dir[direction
                                    ],
                                                            Or(self.
                                                                conveyor[x
                                                                ][y] !=
                                                                self.
                                                                direction
                                                                [0],
                                                                self.
                                                                    assembler
                                                                    [x][y]
                                                                    != 0),
                                                            False))
                    inserter_input.append(If(self.inserter[i][j] != self.
                        direction[0], Or(direction_clauses), True))

        return inserter_input
```

```
    def inserter_output(self):
        inserter_output = []

        for i in range(self.height):
            for j in range(self.width):
                if not self.is_output(i, j):
                    direction_clauses = []
```

```
                for direction in range(1, self.n_dir):
                    x, y = i + self.dir_shift[direction][0], j + self
                        .dir_shift[direction][1]
                    if 0 <= x < self.height and 0 <= y < self.width:
                        # Inserter can output to a conveyor or an
                            assembler
                        direction_clauses.append(If(self.inserter[i][
                            j] == self.direction[direction],
                                                    Or(And(self.
                                                        conveyor[x][y]
                                                         != self.
                                                        direction[0],
                                                        self.conveyor[
                                                            x][y] !=
                                                            self.
                                                            opposite_dir
                                                            [direction
                                                            ]),
                                                       self.assembler[x
                                                           ][y] != 0),
                                                    False))
                inserter_output.append(If(self.inserter[i][j] != self
                    .direction[0], Or(direction_clauses), True))

        return inserter_output
```

Amb totes aquestes restriccions ja queden definides les rutes, tot seguit alguns exemples d'instàncies:

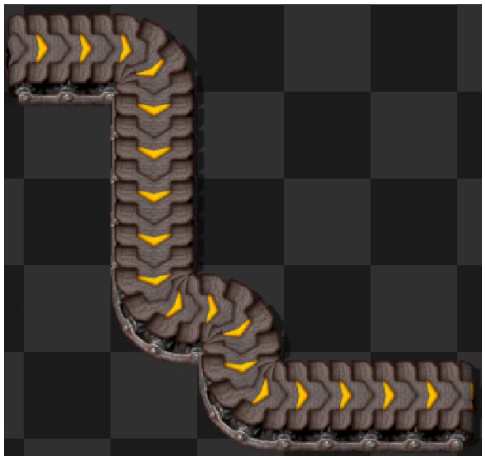

Figura 9: Representació gràfica de la variable `conveyor`

1	6	0	0	0
0	8	0	0	0
0	13	0	0	0
0	18	19	0	0
0	0	21	22	23

Figura 10: Model de la variable `route`

Figura 11: Representació gràfica de la variable `conveyor`

1	2	22	23	24
0	3	0	0	0
0	4	0	0	0
0	7	0	0	0
14	12	15	20	23

Figura 12: Model de la variable `route`

2.2 Restriccions dels assembladors

A l'hora de representar els *assemblers* hem de tenir en compte que ocupen un espai de 3x3 caselles, això complica la detecció de col·lisions entre ells i entre la resta d'objectes, així doncs, s'han usat dues variables per poder tenir control sobre la posició i àrea que ocupa cada *assembler*. La primera variable `assembler` guarda la posició del centre del *assembler* i tracta d'una matriu de mida $width-2 \times height-2$ amb domini $[0..(width/3)\ times(height/3)]$. El motiu pel qual la mida de la matriu no és la del *blueprint* és perquè els *assemblers* en ocupar 3x3 caselles sabem que no hi ha cap assignació al perímetre exterior que sigui vàlida, així que ignorem aquestes posicions. La segona variable `collision` és una matriu de mida $width \times height$ i domini $[0..(width/3) \times (height/3)]$. Aquesta variable guarda en quines posicions hi ha un *assembler*, tot seguit les restriccions que en creen amb aquestes variables que ens permeten el control sobre les colisions.

2.2.1 Set collision

Aquesta restricció s'assegura que per cada posició de la matriu de la variable `assembler` on el valor sigui superior a 0 (és a dir que hi ha un *assembler*), les 3x3 caselles adjacents a la matriu de la variable `collision` hagin de ser del mateix valor. D'aquesta manera ja tenim una variable on tenim representades les caselles on hi ha *assemblers*. Amb aquesta restricció ja ens assegurem que no hi hagi col·lisions entre *assemblers*, ja que en cas que el *solver* hagi assignat dos *assemblers* a posicions on intersequen, aquesta restricció farà que hi hagi caselles que han de prendre dos valors diferents, avaluant així a fals i sent necessari la reubicació de les posicions per evitar-ho. Així i tot, amb això no n'hi ha prou, ja que no estem forçant a les caselles on no hi ha *assemblers* que prenguin el valor 0, impedint així la utilització d'aquesta variable per saber on hi ha exactament un **assembler**. Per arreglar aquest comportament necessitem la següent restricció:

0	0	0
0	0	0
0	0	0

Figura 13: Variable `assembler` sense cap *assembler* present

0	0	0	0	0
0	1	1	0	0
1	0	0	0	0
1	1	0	1	0
0	0	1	0	0

Figura 14: Variable `collision` prenent valors > 0 on no hi ha assemblers

2.2.2 Link assembler collision

Amb aquesta restricció hem d'aconseguir que `collision` nomes pregui valors > 0 on hi ha un assembler, per a fer-ho nomes cal assegurar que per cada posicio de la matriu de la variable `collision` si hi ha una casella que ha pres valor > 0 llavors una de les 8 caselles adjacents a la variable`assembler` ha de tenir un valor > 0.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	4
0	0	0	0	0	0
0	0	0	0	0	0

Figura 15: Variable `assembler` amb 2 assignacions > 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	1	1	0	4	4	4
0	1	1	1	0	4	4	4
0	1	1	1	0	4	4	4
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figura 16: Variable `collision` prenent valors a l'àrea 3x3 només on hi ha presents *assemblers*