Projet *Puyo*

Ce sujet est composé de quatre parties. La section 1 décrit brièvement le jeu et le déroulement d'un tour. La section 2 décrit en détail toutes les règles du jeu. La section 3 décrit la base projet lui-même : ce que vous devez réaliser pour avoir une note correcte. Enfin, la section 4 décrit les extensions possibles pour avoir la note la plus haute possible.

1. Description du jeu

Puyo-Puyo est une série de jeux-vidéos développés par Compile depuis 1991 sur de nombreuses plateformes. C'est un jeu de réflexion et de réflexes similaire à Tetris, joué normalement en versus. L'objectif de ce projet est d'implanter une version **simplifiée** de *Puyo-Puyo*, décrite dans les sections qui suivent.

Les règles du jeu. Les éléments de base du jeu sont les *puyos*. Ce sont de petites boules colorées pouvant avoir 5 couleurs différentes : rouge, vert, jaune, bleu et magenta.

Figure 1. La grande famille des *puyos*

Le terrain de jeu est une grille faisant 6 cases de largeur et 12 cases de hauteur.
Notre version simplifiée de *Puyo-Puyo* se joue en tour par tour et avec un seul joueur.
Chaque tour de jeu est constitué des étapes suivantes :

(1) Le jeu propose au joueur une *pièce* constituée de deux *puyos* adjacents à la couleur arbitraire.

(2) Le joueur peut faire tourner la pièce comme il le souhaite (voir Rotation et placement).

(3) Le joueur choisit dans quelle colonne de la grille il souhaite placer la pièce.

(4) La pièce est alors placée tout en haut de la grille dans la colonne indiquée.

(5) La pièce descend dans la grille selon les règles de la gravité (voir Gravité).

(6) Si quatre *puyos* ou plus sont adjacents (voir Règles de détection), le groupe est supprimé de la grille et des points sont attribués au joueur (voir Calcul du score). La gravité s'applique alors sur la grille.

(7) On répète l'étape précédente jusqu'à ce qu'il n'y ait plus de groupes de *puyos* à supprimer sur la grille (voir Chaînes de combos).

La figure 2 montre les 4 premiers tours de jeu d'une partie. [1]

But du jeu. Marquer le plus de points possibles sans perdre. Le joueur perd lorsqu'il tente de placer une nouvelle pièce dans une colonne déjà remplie.

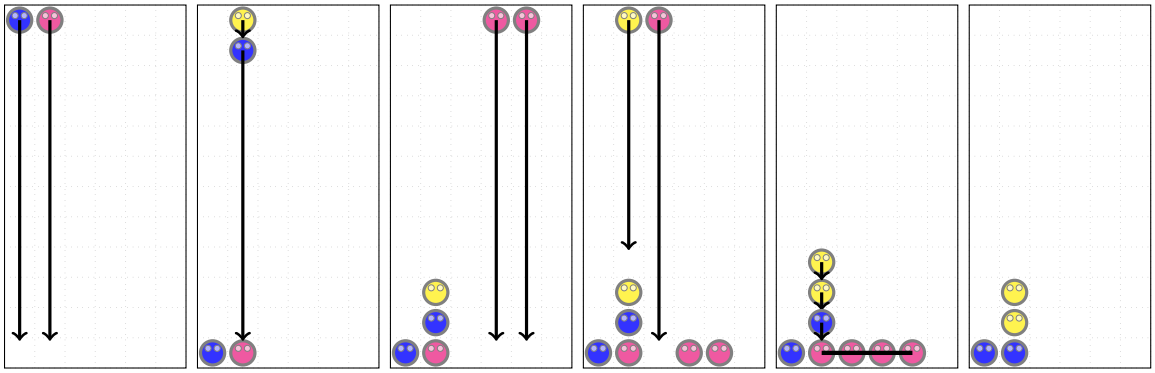

FIGURE 2. Exemple des 4 premiers coups d'une partie, d'une ligne et de l'action de la gravité

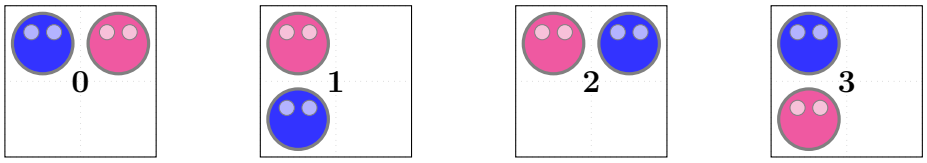

FIGURE 3. Rotations (0, 1, 2, 3) d'une pièce placée dans la première colonne

2. Détail des règles

Rotation et placement. Une pièce peut avoir 4 rotations différentes numérotées de 0 à 3, indiquées dans la figure 3.

Par défaut, une pièce est toujours donnée en rotation 0. Au moment de placer la pièce tournée, on considère que la colonne choisie par le joueur est celle dans laquelle on va placer le *puyo* **le plus à gauche** de la pièce.

Par exemple, dans la figure 2, le joueur a choisi comme colonnes successives 1, 2, 4 et 2 (les colonnes étant numérotées de 1 à 6).

Gravité. Comme indiqué plus haut, la gravité s'applique à deux moments d'un tour : lorsqu'une nouvelle pièce est placée (pour qu'elle descende), et lorsqu'un groupe est supprimé (par exemple dans la cinquième grille de la figure 2).

Lorsque la gravité s'applique à la grille, tout *puyo* étant au dessus d'un certain nombre de cases vides doit descendre dans sa colonne jusqu'à rencontrer une case pleine ou le bas de la grille.

Les deux *puyos* d'une pièce ne sont pas solidaires : ils peuvent se séparer et descendre à des niveaux différents. C'est par exemple le cas de quatrième grille de la figure 2.

Règles de détection. Il y a deux règles possibles pour la détection des groupes de *puyos*, que l'on numérote 1 et 2.

Règle 1 Dans la version simplifiée de notre jeu, on considère que quatre *puyos* (ou plus) forment un groupe à supprimer lorsqu'ils sont adjacents et sur la même colonne ou la même ligne. Ce comportement est représenté dans la figure 4 où 3 groupes sont détectés. On note que si un même *puyo* fait partie de deux groupes (comme celui en bas à gauche de la grille), alors il est compté **dans les deux groupes**.

Règle 2 Dans le jeu original, quatre *puyos* (ou plus) adjacents sont considérés comme un groupe, peu importe la forme du groupe. Elle est illustrée dans la figure 5.

1. Vous pouvez aussi regarder à quoi ressemble le jeu original pour vous faire une idée des règles. Pour rappel, notre version simplifiée sera en tour par tour, et en solo.

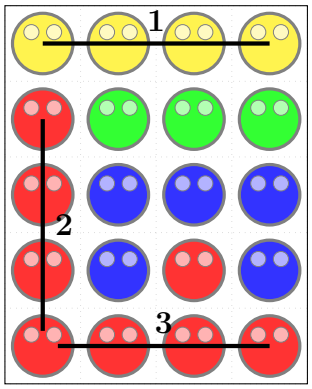

FIGURE 4. Règle 1

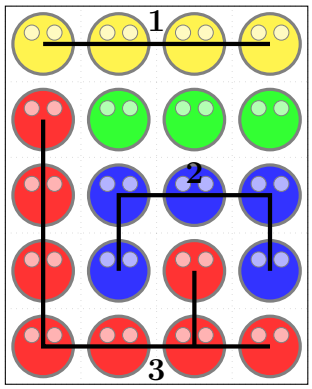

FIGURE 5. Règle 2

Chaînes de combos. Dans le jeu original comme dans notre version, il y a un mécanisme de *combos*, qui est d'ailleurs la meilleure manière de marquer de nombreux points, sachant qu'on peut enchaîner les combos sans limite (dans cette vidéo, la règle 2 est utilisée).

Lorsque dans un tour de jeu un groupe de *puyos* est supprimé suite à l'action du joueur, la nouvelle grille résultant de l'action de la gravité peut à nouveau contenir un groupe, qui va lui-même être supprimé, et ainsi de suite.

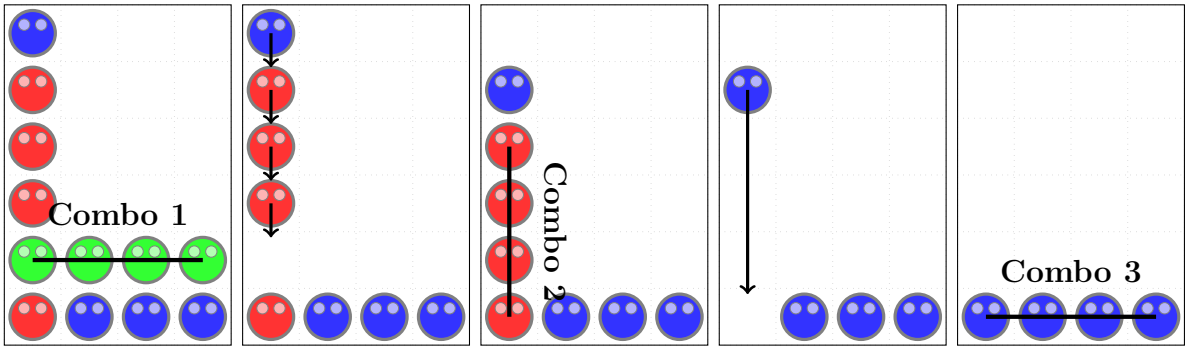

FIGURE 6. Exemple de trois combos enchaînés

Dans la figure 6 par exemple, il y a une chaîne de 3 combos : trois groupes sont formés juste par l'action de la gravité, sans intervention du joueur.

Calcul du score. Des points sont ajoutés au joueur à chaque fois qu'un (ou plusieurs) groupe(s) de *puyos* est supprimé. On fixe le **score maximal** à 999999 points [2]. Les points à ajouter sont calculés selon la formule suivante et ne doivent eux-mêmes pas dépasser ce seuil.

$$Points = 10 \times \texttt{puyos} \times (4^{\texttt{combos}} + 3^{\texttt{groupes}})$$

où :

— `puyos` correspond au nombre de *puyos* supprimés en tout (au moins 4, donc)

— `combos` correspond au numéro du combo. Il est donc égal à 1 si la suppression résulte d'une action du joueur, et augmente pour chaque suppression due à la gravité (voir la figure 6 par exemple, où ce nombre vaut successivement 1, 2 puis 3).

— `groupes` correspond au nombre de groupes supprimés d'un coup. Par exemple, dans les figures 4 et 5, ce nombre vaut 3.

2. Tout score supérieur doit être ramené à 999999.

Quelques exemples :
— sur la figure 2, un seul groupe de 4 est supprimé, sans provoquer de combos. Le joueur gagne donc $10 \times 4 \times (4^1 + 3^1) = 280$ points. C'est la plus faible valeur que l'on puisse marquer.
— sur la figure 4, trois groupes sont supprimés d'un coup, sans provoquer de combos. Au total, 12 *puyos* sont supprimés (l'un d'eux est compté deux fois). On a donc $10 \times 12 \times (4^1 + 3^3) = 3720$ points.
— de manière similaire, la figure 5 rapporte $10 \times 17 \times (4^1 + 3^3) = 5270$ points.
— dans la figure 6, 3 combos successives sont réalisés, où à chaque fois un seul groupe de 4 *puyos* est supprimé. On applique donc trois fois la formule, une fois pour chaque combo :

$$10 \times 4 \times (4^1 + 3^1) + 10 \times 4 \times (4^2 + 3^1) + 10 \times 4 \times (4^3 + 3^1) = 3720 \text{ points}$$

3. Description du projet

Le projet consiste à développer un programme supportant trois modes décrits plus loin : *Simulation*, *Solo* et *IA*. Dans tous les cas, votre programme doit pouvoir être lancé en exécutant `./puyo FICHIER` où `FICHIER` est le nom d'un fichier d'entrée. La première ligne de ce fichier décrit le mode dans lequel on veut jouer, et la suite du fichier dépend du mode.

Pour **avoir 12**, vous devez respectez les critères suivants :
— Les fichiers de votre projet compilent
— Votre programme supporte le Mode simulation correctement, avec la **règle 1**
— Toutes les fonctions sont *spécifiées*, *documentées* et *testées*
— La compilation doit être séparée, c'est à dire que votre programme doit être divisé en plusieurs fichiers comme ce qui a été vu dans le cours
— La soutenance est soignée
— Le rapport est bien rédigé (orthographe,...) et comprend :
  (1) Une présentation de votre programme (qu'est-ce qui a été réalisé dans le sujet),
  (2) Une explication de la structure de votre code, et des fonctions principales.
  (3) Une explication sur comment le travail a été réparti dans le binôme,
  (4) Les difficultés que vous avez rencontrées, et les problèmes connus de votre programme.

Pour avoir **plus que 12**, vous devrez au choix implanter les deux autres modes de jeu et/ou réaliser certaines des améliorations proposées (ou celles que vous inventez !) sur chacun des modes dans la dernière section. Une fois le mode simulation basique réalisé, vous n'avez aucune obligation de faire les choses dans l'ordre du sujet.

3.1. Mode simulation.

Ce mode consiste à *rejouer* une partie décrite dans le fichier d'entrée. Ce dernier a le format suivant :

```
MODE SIMULATION
PIECE
ROTATION COLONNE
PIECE
ROTATION COLONNE
...
```

Autrement dit : après la première ligne annonçant le mode de jeu, le fichier décrit la **pièce** qui a été proposée au joueur sur une ligne. Sur la ligne suivante, le fichier décrit la **rotation** (entre 0 et 3) que le joueur a choisi pour la pièce et la **colonne** (entre 1 et 6) dans laquelle il l'a placée. Puis on recommence jusqu'à ce qu'il n'y ait plus de pièces.

La pièce est décrite par deux lettres, chacune correspond à un *puyo* en fonction de sa couleur : **R**ouge, **V**ert, **J**aune, **B**leu ou **M**agenta.

Ainsi, le fichier correspondant à la partie en figure 2 pourrait être :

```
MODE SIMULATION
BM
0 1
BJ
1 2
MM
0 4
MJ
2 2
```

Votre programme doit :

(1) Lire ce fichier

(2) En partant d'une grille vide, exécuter autant de tours de jeu que d'actions décrites dans le fichier

(3) À la fin, écrire dans un fichier **sortie.txt** la grille et le score final

Chaque case de la grille est représentée par la lettre du *puyo* qu'elle contient, où un point si elle est vide.

Ainsi le fichier `sortie.txt` correspondant au fichier d'entrée ci-dessus serait :

```
......
......
......
......
......
......
......
......
......
.J....
.J....
BB....
280 points
```

Note. Si une action décrite dans le fichier provoque la défaite du joueur, alors on s'arrête de lire le fichier et on écrit `DEFAITE` dans le fichier de sortie, avant la grille et le score.

Pour vous aider, vous trouverez dans le dossier `exemples/simulation` plusieurs fichiers d'entrée pour tester votre programme ainsi que le fichier de sortie attendu pour chacun de ces tests.

3.2. **Mode solo.** Ce mode consiste à faire jouer une partie à un utilisateur. Le fichier d'entrée doit avoir le format :

```
MODE SOLO
PIECE
PIECE
...
```

Votre programme doit lire les pièces du fichier, et présenter à l'utilisateur une interface lui permettant de placer chacune de ces pièces.

Au minimum, l'interface doit permettre au joueur de choisir pour chaque pièce la rotation et la colonne qu'il souhaite de manière textuelle, et afficher au fur et à mesure le score actuel et l'état de la grille.

3.3. **Mode IA.** Cette fois-ci, c'est à votre programme de décider comment placer les pièces ! Le fichier d'entrée doit avoir le même format que le mode solo (sauf la première ligne qui doit être `MODE IA`).

Votre programme doit alors écrire dans un fichier `sortie.txt` les actions successives qu'il décide de faire, avec le format suivant :

```
ROTATION COLONNE
ROTATION COLONNE
ROTATION COLONNE
...
```

Il doit y avoir autant d'actions que de pièces dans le fichier d'entrée, sauf si une action mène à la défaite, auquel cas on écrit `DEFAITE`.

Dans tous les cas, le fichier doit finir, comme le mode simulation, par la grille finale et le score final.

Au minimum, votre programme doit faire un choix aléatoire parmi les choix disponibles.

À la fin du semestre, nous organiserons un concours entre vos différentes IA, qui seront jugées selon différents critères (temps de survie, nombre maximal de combos, score maximal...). Les résultats seront rendus publics !

4. Extensions possibles

Écrire un programme qui supporte les trois modes basiques vous garantira la note de **14**. Ensuite, de nombreuses améliorations sont possibles pour aller au-délà, sans obligation d'ordre !

(1+ pt) Nous vous donnons quelques fichiers de test pour chacun des modes. Pourquoi ne pas écrire vous-même un petit programme permettant de générer des fichiers supplémentaires pour tester votre programme ? De plus, le mode solo pourrait écrire un fichier `sortie.txt` dans le format du mode simulation, afin qu'une partie jouée puisse être facilement rejouée.

(1 pt) Pourquoi ne pas réutiliser votre jolie interface du mode solo pour les autres modes ? Faire en sorte que si on donne à votre programme un deuxième paramètre `anim`, les parties des modes simulation et IA utilisent votre interface pour afficher le déroulement de la partie.

(4 pts) Tous les modes considèrent que la règle 1 est utilisée pour détecter les groupes. Vous pouvez vous rapprocher du jeu original en implantant plutôt la règle 2. Dans ce cas, tous vos fichiers de sortie devront commencer par `REGLE 2` afin de signaler que cette extension est active.

(2+ pts) Le mode solo et son interface utilisateur permet d'être très créatif sur ce qui peut être amélioré. Voici quelques suggestions, ordonnées par difficulté :

- améliorer l'interface terminal pour le confort du joueur (couleur, animations montrant la chute des pièces et les combos successifs, ...)
- faire une interface graphique, en utilisant la bibliothèque SFML
- ajouter des sons, des musiques, des images, ...
- passer outre l'aspect tour par tour et se rapprocher du jeu original (intégrer une vitesse de chute de la pièce, qu'on peut contrôler et tourner au fur et à mesure qu'elle tombe)
- pourquoi pas un mode versus comme dans le jeu original, où deux joueurs s'affrontent...
- toute idée intéressante que vous pouvez avoir !

(2+ pts) Enfin, toute amélioration du mode IA est bonne à prendre. On peut commencer par une IA qui essaie de ne pas perdre, de marquer le plus de points possibles, ou même de construire les plus grandes chaînes de combos ! Arriver dans le haut du podium en fin de semestre vous garantira quelques points bien mérités...