Université Côte d'Azur
Faculté des Sciences

Rapport :
Projet Pickomino

Projet réalisé par :
Nicolas Zanin
Hugo Savasta

Année universitaire :
2021 / 2022

SOMMAIRE:

Présentation du jeu :

Pickomino est un jeu de dés qui rappelle aux premiers abords une ambiance du style 421 ou Yam's.
Lancez les 8 dés, choisissez une valeur de dés à mettre de côté, puis faites le pari de relancer ou non. Vous pouvez relancer les dés restants autant de fois que vous le souhaitez, à condition de toujours pouvoir retenir une valeur de dés différente. Si vous êtes dans l'incapacité de le faire, alors vous devrez rendre un de vos précieux Pickomino et passer votre tour ! Lorsque votre tour est fini, faites l'addition pour savoir quel Pickomino prendre au centre de la table ou même dans la main de vos adversaires !

Par rapport au jeu traditionnelle, le projet permet en plus :

-Faire sauvegarde en binaire d'une partie en cours
-Lire une sauvegarde / ou faire une nouvelle partie
-Joueur intelligent (joueur capable de prendre des décisions seul)
-Humain vs machine(joueur intelligent)

<u>Stratégie pour faire le projet :</u>

Nous avons réduit le jeu en plusieurs sous problèmes.

- 1ère étape : Tour d'un joueur

Au début, nous avons juste fait en sorte de pouvoir faire jouer 1 joueur :
-Demander nom joueur
-Faire choisir une valeur à mettre de côté
-Supprimer le nombre de dès selon le nombre de valeur correspondant à la valeur choisie par le joueur.
-Puis, lui demander de choisir à nouveau une valeur
-Terminer le tour du joueur s'il décide de s'arrêter ou s'il ne peut plus choisir de valeur.
Ensuite, après que son tour soit terminé, lui faire recommencer un tour avec les mêmes étapes.
Une fois cela opérationnel, faire prendre au joueur un pickomino au centre de la table s'il a gagné en fonction de la somme de ses dés ou bien de rendre un pickomino s'il a perdu.

- 2ème étape : Passer au joueur d'après

Passer du joueur 1 au joueur d'après et ainsi de-suite pour faire un tour complet de tous les joueurs .

- 3ème étape : Prendre pickomino à un joueur

Pour ceci, nous avons du créer une fonction cherchant si le sommet de la pile d'un autre joueur correspond à la somme des dés du joueur actuel.

- 4éme étape : Joueur Intelligent

Cette étape a été assez difficile par rapport au fait de trouver une stratégie

qui permet à ce joueur intelligent de faire les bons choix et de gagner. (difficulté pour trouver la stratégie nous avons du faire une trentaine de parties pour trouver la bonne...).
Ensuite, nous avons du créer une fonction permettant que le joueur intelligent puisse continuer son tour comme un joueur humain, l'IA peut également, comme un joueur humain, décider de s'arrêter.Toutefois, ce jeu reste un jeu de hasard, il se peut que même un joueur intelligent n'ait pas de chance sur le tirage des dès et puisse perdre un tour.

- 5éme étape : Sauvegarde (bonus)

Cette étape était notre avant-dernier objectif, nous avons fait une sauvegarde en binaire qui, à la fin du tour de tous les joueurs, sauvegarde toute seule. Par exemple, lorsque 3joueurs ont fini de jouer et que le tour revient au joueur 1, alors la sauvegarde s'effectue toute seule.

- 6éme étape : Le Visuel

-Demande si on a une sauvegarde :
    -Si oui, la continuer ou pas
    -Sinon, on demande que l'utilisateur rentre un nom de fichier(exemple : SAVE) qui sera complété automatiquement par un « .bin» grâce à une fonction.

-Nombre Joueurs
On demande le nombre de joueur puis le nombre d'humain, le nombre de joueur intelligent sera calculé en soustrayant le nombre d'humains au nombre de joueurs.
Ensuite, on demande le nom des humains, puis, des IA afin que chaque joueur ait un surnom dans le jeu.
Une fois les noms des joueurs rentrés, on affiche chaque joueur et son numéro avec le surnom qui lui est associé.

-Pickomino
En effet, nous avons créer une fonction dessinant les pickominos visibles au centre de la table à l'aide de caractère ASCII. A chaque début de tour d'un joueur, elle affiche les pickominos disponibles et ceux qui ne sont

plus disponibles sont représentés par des « # » comme si le domino était retourné.

Description du code :

- Initialisation :

-void initStruct(Player *player, short indexplayer);
Cette fonction initialise la structure du joueur selon son indice.
Player *player : correspond à la structure du joueur
short indexplayer : indice joueur

-short fillStruct(Player *player);
Cette fonction remplit la structure.
Player *player : correspond structure

-void initValue(short *t, short start, short number, short value);
Cette fonction remplit le tableau des valeurs mises de côtés par le joueur.
short *t : pointeur vers un tableau
short start :indice du début
short number : nombre dés
short value : correspond valeur choisit par utilisateur

-void rollDice(short *t, short number);
Cette fonction sert à lancé les dés qu'il reste.
short *t : pointeur vers un tableau qui prends les dés.
short number : nombre de dés

- Verifications:

-short numberOccurence(short *t,short value, short end);
Cette fonction compte le nombre de fois que la valeur choisit apparaît pour le joueur intelligent.
short *t : pointeur vers un tableau contenant les dés
short value : valeur choisie par joueur intelligent
short end : nombre de dés qui sert comme indice

-short checkValue(short *tab, short value, short end);
Cette fonction vérifie si la valeur choisie est disponible.
short *tab : pointeur vers tableau de valeurs mises de côtés
short value : valeur saisie par le joueur /joueur intelligent
short end : nombre de dés qui sert comme indice de remplissage

-short checkPossibility(short *rolldice, short *dicekeep, short end, short numberdice);
Cette fonction vérifie s'il reste des valeurs disponibles pour le joueur.
short *rolldice : pointeur vers tableau des dés
short *dicekeep : pointeur vers tableau des valeurs de dés saisies
short end : nombre de dés
short numberdice : nombre dés restants

-short checkStack(Player *player, short numberplayer, short indexplayer, short sumdice);
Cette fonction vérifie si le sommet de la pie d'un joueur correspond à la somme des dés du joueur actuel.
Player *player : correspond structure du joueur
short numberplayer : correspond nombre de joueurs
short indexplayer : correspond indice du joueur
short sumdice : correspond somme dés du joueur

-short checkPickominoAvailable(short *pickomino, short number);
Cette fonction cherche si il existe un pickomino disponible.
short *pickomino : pointeur vers tableau des pickominos
short number : correspond numéro pickomino

-short checkLastElPlayer(Player *player, short sumdice, short indcurrentplayer, short numberofplayer);
Cette fonction vérifie si le sommet de la pile d'un joueur est égal a la somme des dés.
Player *player : correspond structure joueur
short sumdice : correspond somme des dés
short indcurrentplayer : indice du joueur actuel
short numberofplayer : nombre de joueur total

-short checkSumDice(Player *player, short *pickomino, short numberplayer, short indcurrentplayer, short sumdice);
Verifie si la somme des dés est bien entre 21 et 36 et par la suite il faudra qu'elle verifie si la valeur de la somme correspond à une valeur de pickomino disponible.
Player *player : correspond structure joueur
short *pickomino : pointeur vers tableau des pickominos
short numberplayer : nombre de joueur total
short indcurrentplayer : indice joueur actuel
short sumdice : somme des dés

- Prendre ou rendre pickomino :

-short getNewSum(Player *player, short *pickomino, short numberplayer, short indcurrentPlayer, short sumdice);
Cette fonction vérifie si le sommet de la pile d'un joueur est égal a la somme des dés et permet de regarder si dans le tableau des pickominos, il y a le pickomino disponible.
Player *player : structure joueur
short numberplayer : nombre joueurs total
short indcurrentplayer : indice joueur actuel
short sumdice : somme des dés

-void replaceDomino(Player *player, short *pickomino, short indexPlayer);
Cette fonction replace le pickomino au centre de la table.
Player *player : correspond structure joueur
short *pickomino : pointeur vers tableau pickomino
short indexPlayer : indice joueur

-void lostDomino(Player *player, short *pickomino, short *domino, short indexplayer);
Cette fonction appelle la fonction replace domino afin de remettre au centre de la table le domino.
Player *player : correspond structure joueur
short *pickomino : pointeur tableau pickominos
short *domino : nombre de pickominos
short indexplayer : indice joueur

-short getDominos(Player *player, short *pickomino, short *domino, short sumdice, short numberplayer, short indexplayer);
Cette fonction permet de prendre un pickomino au centre de la table ou à un joueur.
Player *player : correspond stucture joueur
short *pickomino : pointeur vers tableau pickominos
short *domino : nombre pickominos
short sumdice : somme des dés
short numberplayer : nombre total joueur
short indexplayer : indice joueur actuel

- Choix humain/Ordi :

-void choice(short *rolldice, short *dicekeep, short *numberdice, short *sumdice);
Cette fonction permet à un joueur humain de faire un choix.
short *rolldice : pointeur vers tableau des dés lancés
short *dicekeep : pointeur vers valeurs mises de côtés
short *numberdice : pointeur vers nombre de dés restants
short *sumdice : pointeur vers somme de dés

-void continuTower(Player *player, short *pickomino, short *domino,short *rolldice,short *dicekeep, short numberplayer, short indexplayer);
Cette fonction permet de continuer le tour d'un joueur humain.
Player *player : correspond structure player
short *pickomino : pointeur vers tableau pickominos
short *domino : nombre pickominos
short *rolldice : pointeur vers tableau des dés lancés
short *dicekeep : pointeur vers valeurs mises de côtés
short numberplayer : nombre total joueur
short indexplayer : indice joueur actuel

-void choiceIA(short *rolldice,short *dicekeep,short *numberdice, short *sumdice,short *state);
Cette fonction permet à un joueur intelligent de faire un choix.
short *rolldice : pointeur vers tableau des dés lancés
short *dicekeep : pointeur vers valeurs mises de côtés
short *numberdice : pointeur vers nombre de dés restants

short *sumdice : pointeur vers somme de dés
short *state : si la valeur 'V' a été choisie

-void continuTowerIA(Player *player, short *pickomino, short *domino, short *rolldice, short *dicekeep, short numberplayer, short indexplayer);
Cette fonction permet de continuer le tour d'un joueur intelligent.
Player *player : correspond structure player
short *pickomino : pointeur vers tableau pickominos
short *domino : nombre pickominos
short *rolldice : pointeur vers tableau des dés lancés
short *dicekeep : pointeur vers valeurs mises de côtés
short numberplayer : nombre total joueur
short indexplayer : indice joueur actuel

- Departager Joueur :

-void score(Player *player, short numberplayer);
Cette fonction calcule le score.
Player *player : correspond structure joueur
short numberplayer : nombre total joueur

-short egality(Player *player, short numerowinner, short numplayeregality);
Cette fonction départage les joueurs en cas d'égalité.
Player *player : correspond structure joueur
short numerowinner : indice du joueur avec le plus grand nombre de pts
short numplayeregality : indice du joueur avec le même nombre de pts

-void winnerGame(Player *player, short numberplayer);
Cette fonction désigne le joueur gagnant.
Player *player : correspond structure du joueur
short numberplayer : nombre total joueur

- Sauvegarde :

-void saveTowerBin(Player *player, char *namefile, short *pickomino, short domino, short numberplayer);
Cette fonction sauvegarde en binaire au début de chaque tour du joueur

numéro1.
Player *player : strcuture joueur
char *namefile : nom de fichier de sauvegarde
short *pickomino : pointeur vers tableau des pickominos
short domino : nombre de domino
short numberplayer : nombre total joueur

-void savePartybin(char *namefile);
Cette fonction créer le fichier de sauvegarde binaire.
Char *namefile : nom de fichier de sauvegarde

-void readSaveBinary(Player *player, char *namefile, short *save, short *pickomino, short *domino, short *numberplayer);
Cette fonction lit le fichier de sauvegarde binaire.
Player *player : correspond structure joueur
char *namefile : nom fichier sauvegarde
short *save : 1 si sauvegarde existe, 0 si sauvegarde n'existe pas
short *pickomino : pointeur vers tableau pickominos
short *domino : pointeur nombre de domino
short *numberplayer : pointeur vers nombre total de joueur

- Visuel :

-void printName(Player *player, short i);
Cette fonction affiche le nom du joueur.
Player *player : correspond structure joueur
short i : indice du joueur

-void printTab(short *t, short number);
Cette fonction affiche un tableau.
short *t : pointeur vers tableau
short number : nombre de case à afficher

-void printStruct(Player *player, short indexplayer, short number);
Cette fonction affiche la pile du joueur.
Player *player : structure joueur
short indexplayer : indice joueur
short number : nombre éléments de la pile

-void drawAscii(char a, char b, char c, char d, char e, char f, short *pickomino, short i);
Cette fonction dessine les pickominos.
Char a,b,c,d,e,f : correspond à une valeur ascii.

-void drawDomino(short *pickomino);
Cette fonction dessine les pickominos.
short *pickomino : pointeur vers tableau des pickominos.

-void drawAsciiUnix(short *pickomino, char *c, short i);
Cette fonction dessine les pickominos.
short *pickomino : pointeur vers tableau des pickominos.
char *c : chaine caractère dessinant le pickomino
short i : entier qui correspond indice

-void drawDominoUnix(short *pickomino);
Cette fonction dessine les pickominos.
short *pickomino : pointeur vers tableau des pickominos.

- Fin Partie :

-void continuParty(Player *player,char *namefile,short numberplayer,short *pickomino,short *domino,short *rolldice, short *dicekeep,short ascii);
Cette fonction permet de continuer la partie tant qu'elle n'est pas finit.
Player *player : structure joueur
char *namefile : nom fichier sauvegarde
short numberplayer : nombre total joueur
short *pickomino : pointeur vers tableau pickominos
short *domino : nombre dominos
short *dicekeep : valeur mises de côtés
short ascii : 1 si système Unix ou MacOS et 0 si système Windows

Problèmes rencontrés et solutions trouvés :

- Prendre le pickomino chez un autre joueur :

solution → créer une fonction checkLastElPayer() et dans la structure stocker l'indice du dernier élément de la pile d'un joueur (laststackelement).

- Joueur intelligent : difficulté pour trouver la stratégie car c'est un jeu de hasard et que l'on ne peut pas vraiment prévoir de véritable stratégie gagnante.

Stratégie : Prendre en priorité un V, puis faire une grosse somme.

Bonus : Sauvegarde Binaire

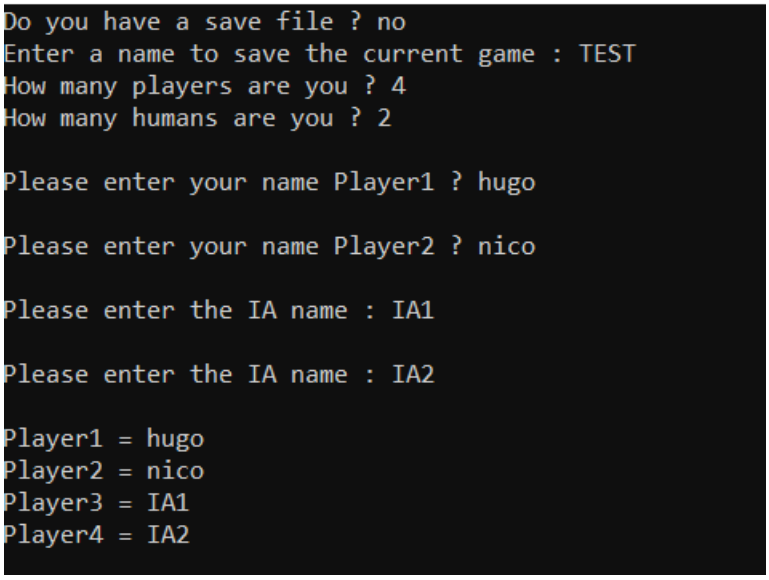
Dans un premier temps, on écrit en binaire dans le fichier avec la fonction 'fwrite()' le nombre de joueurs, nombre de pickominos, la liste des pickominos, et les structures des joueurs.
Puis on ferme le fichier avec le fichier avec la fonction 'fclose()'.

Pour lire la sauvegarde, on ouvre le fichier le fichier en mode lecture.
On prend le nombre de joueur, le nombre de pickominos, la liste des pickominos, et grâce au nombre de joueur on peut prendre le nombre de structure.

La sauvegarde se supprime toute seule lorsque la partie se termine grâce à la fonction remove(namefile).

Cas de Tests :

-Demande de Sauvegarde, du nombre de joueur, noms des joueurs :



-Tour d'un joueur :

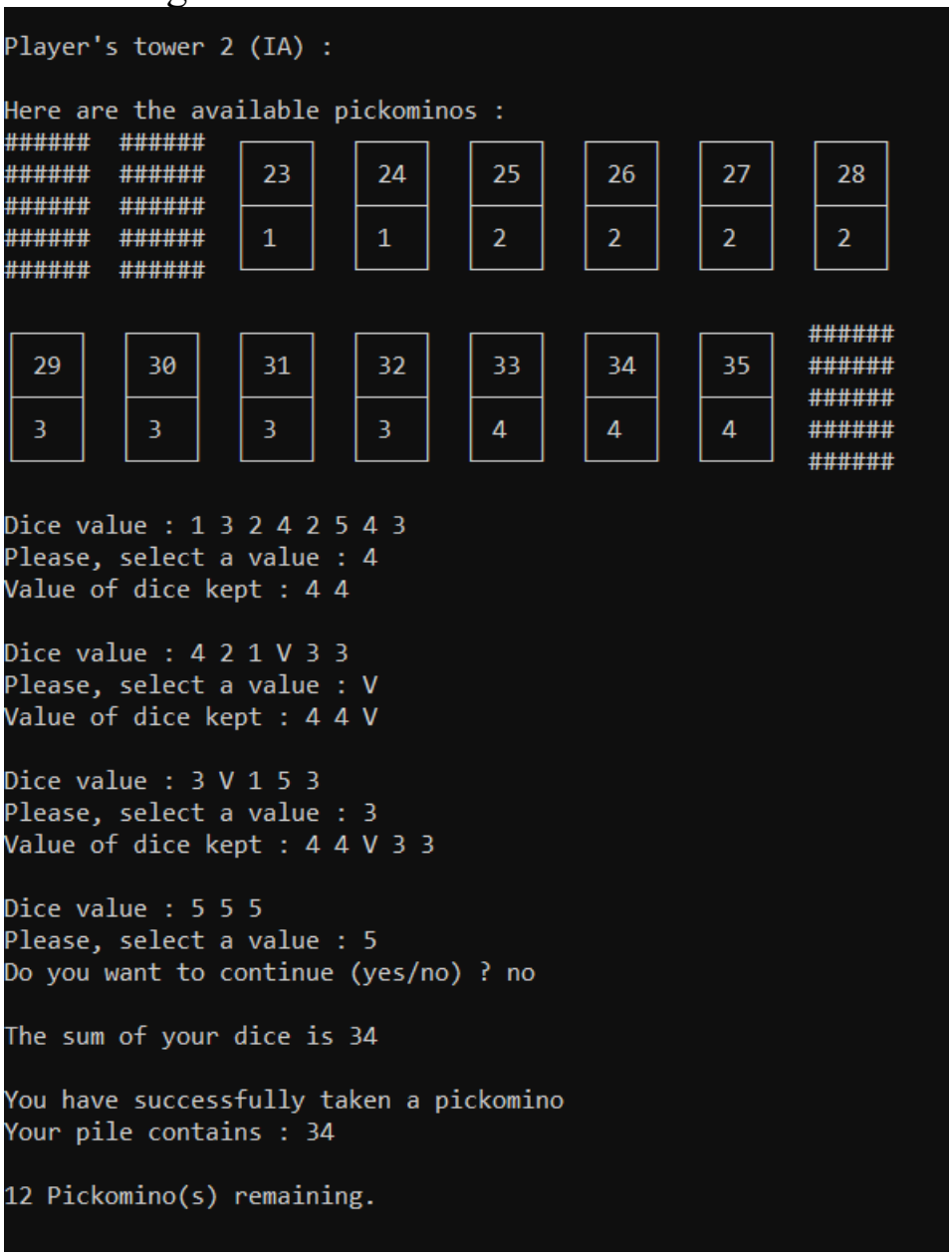
```
Player's tower 1 (hugo) :

Here are the available pickominos :
```

21	22	23	24	25	26	27	28
1	1	1	1	2	2	2	2

29	30	31	32	33	34	35	36
3	3	3	3	4	4	4	4

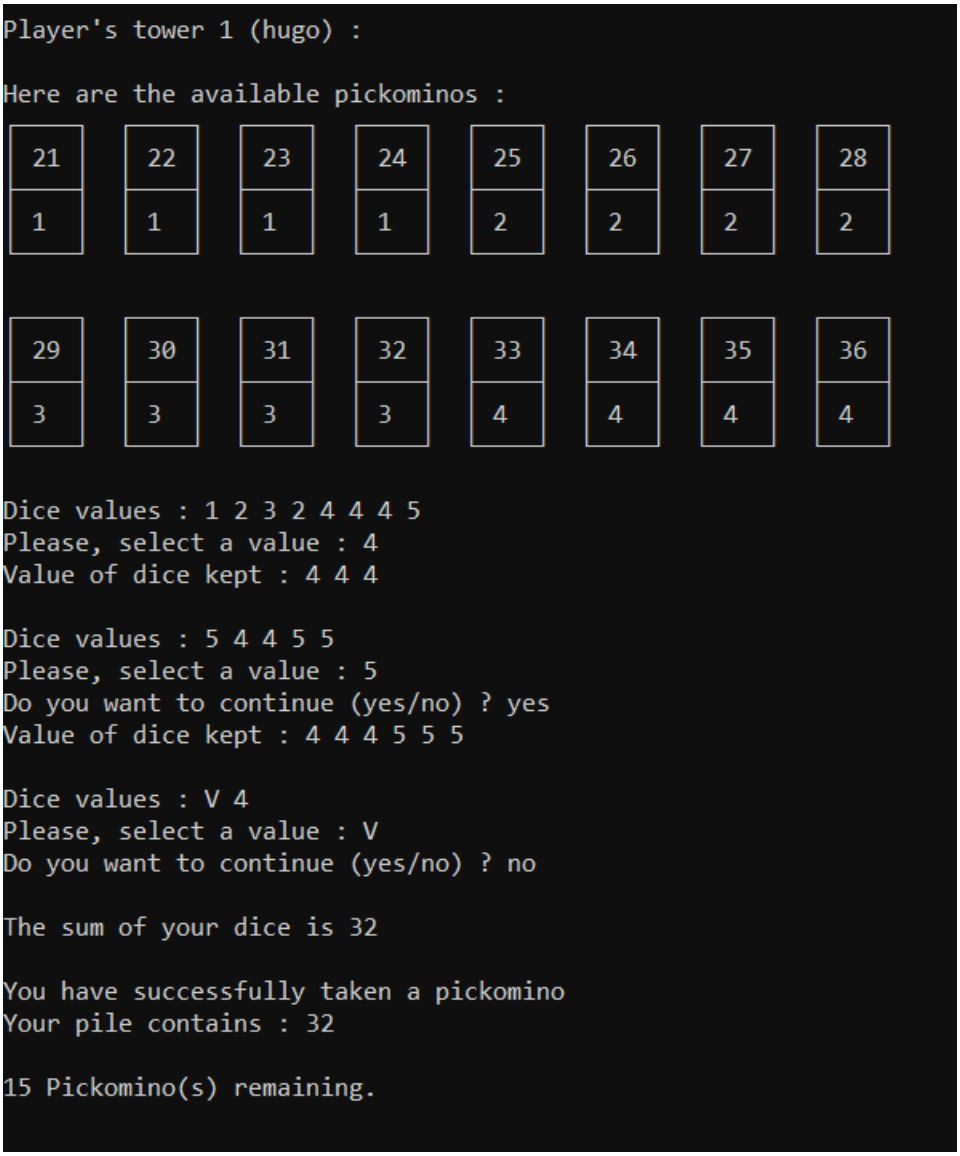
```
Dice values : 1 2 3 2 4 4 4 5
Please, select a value : 4
Value of dice kept : 4 4 4

Dice values : 5 4 4 5 5
Please, select a value : 5
Do you want to continue (yes/no) ? yes
Value of dice kept : 4 4 4 5 5 5

Dice values : V 4
Please, select a value : V
Do you want to continue (yes/no) ? no

The sum of your dice is 32

You have successfully taken a pickomino
Your pile contains : 32

15 Pickomino(s) remaining.
```

-Tour Joueur Intelligent :

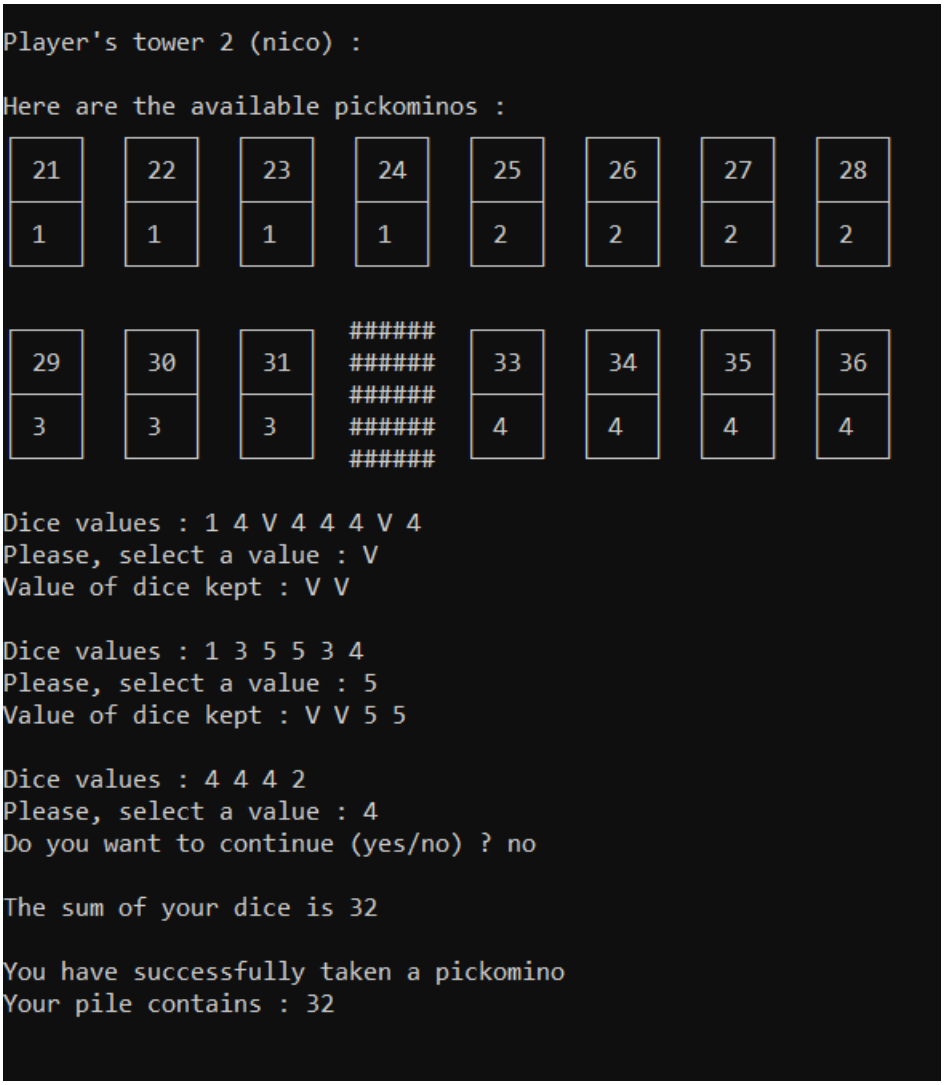


-Prise pickomino à un autre joueur : (2images)

Cette image correspond au joueur prenant un pickomino au centre de la table.



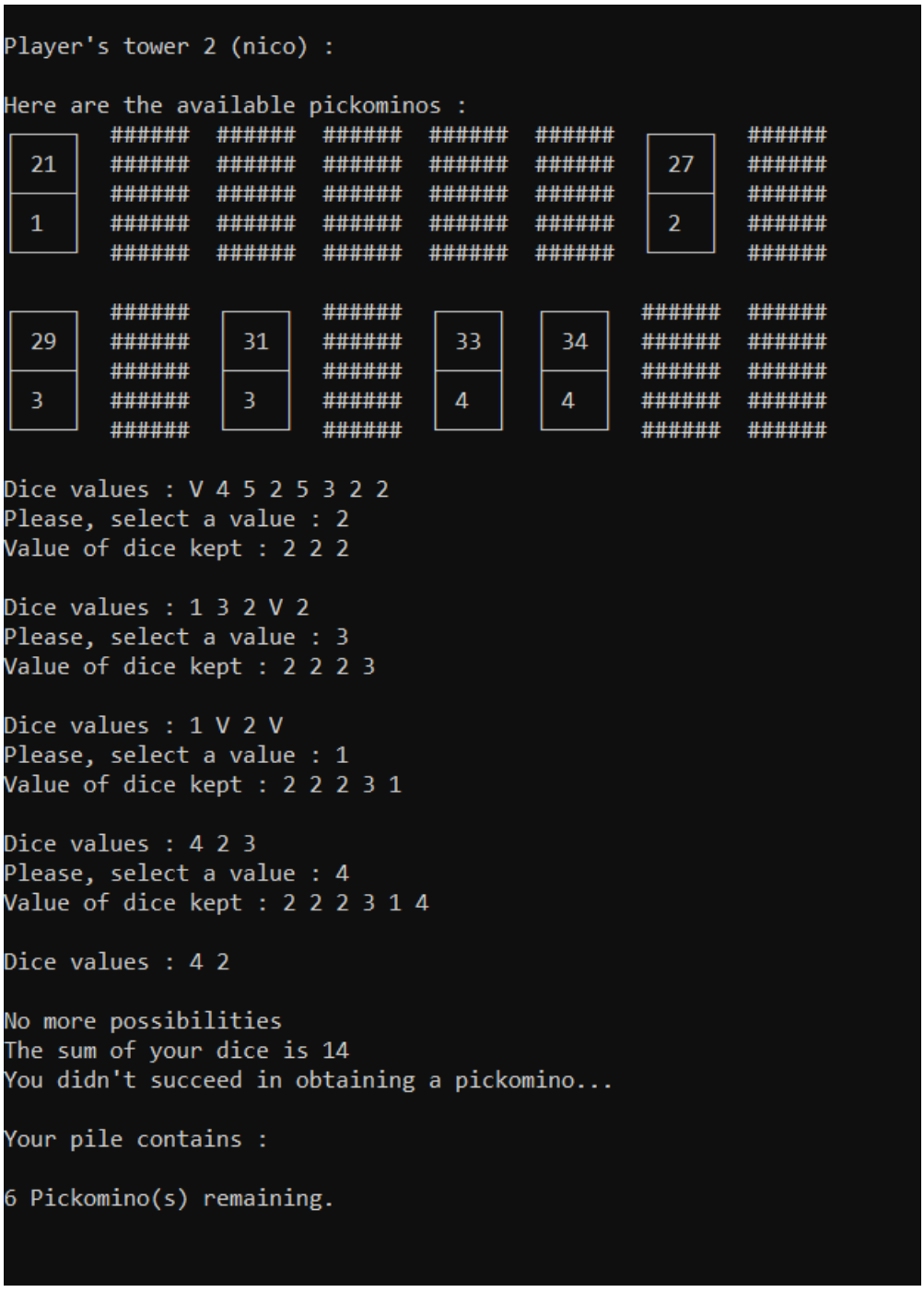
Cette image correspond au joueur prenant le pickomino numéro 32 au joueur adverse.



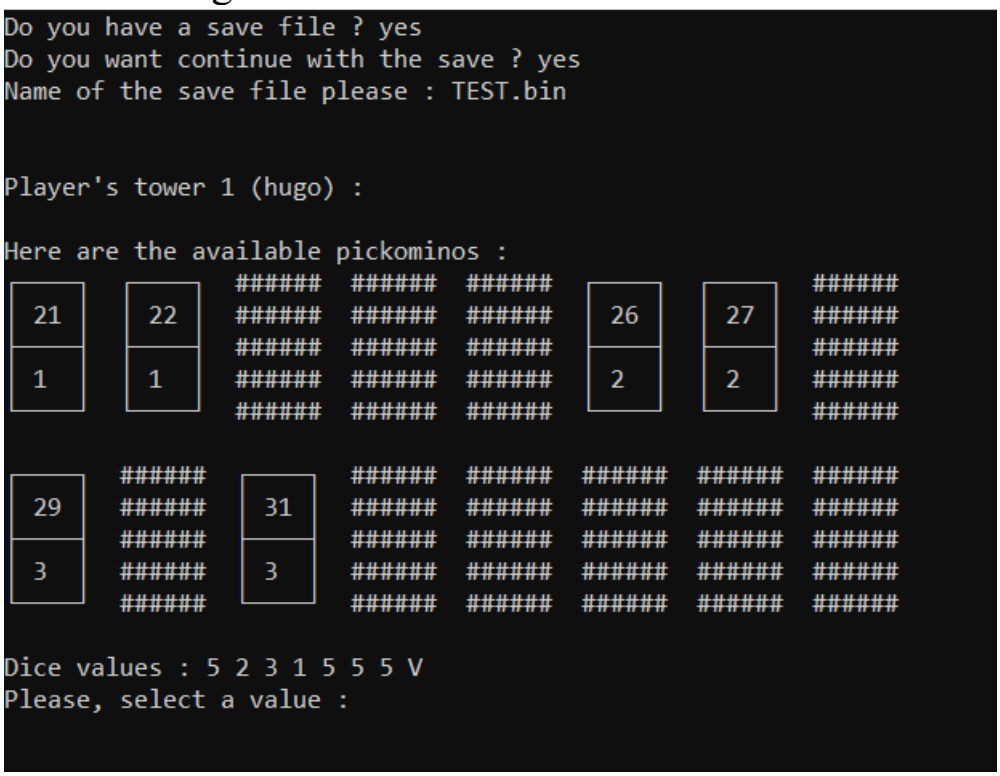
-Tour perdu : (2images)

Cette image correspond au tour avant de perdre, l'image suivante montrera ce qui se passe lorsque l'on perd le tour.

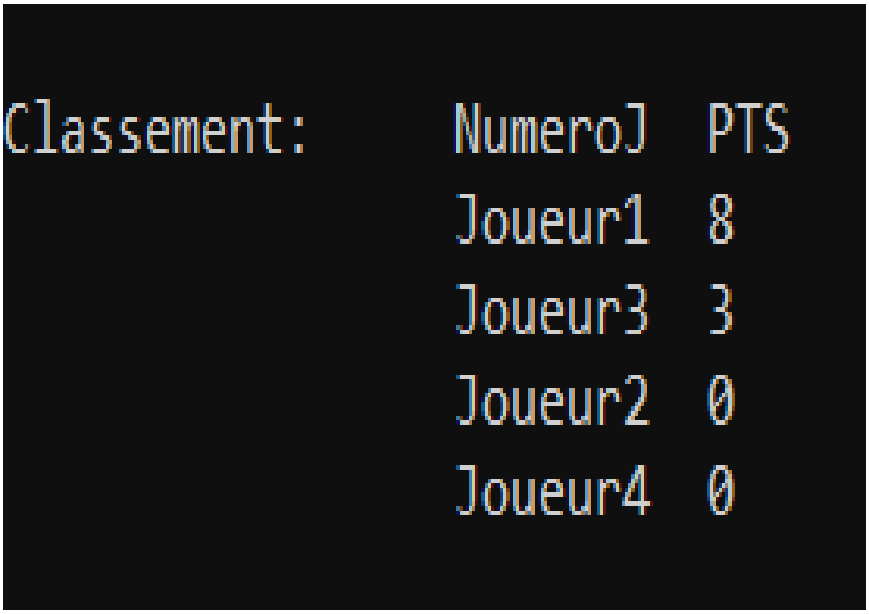
Cette image montre que la pile est vide. Le joueur a replacé son pickomino au centre de la table, il est donc par conséquent plus dans sa pile.



-Reprise d'une sauvegarde :



-Fin de partie :



Conclusion :

Ce projet fut intéressant, cependant par moment il a été compliqué. Cela nous a permis d'apprendre à mieux débuguer notre code ainsi que de mieux l'organiser. Ce projet représente notre premier jeu en langage C, il nous a permit d'apprendre de nombreuses notions qui pourront nous être utiles plus tard dans notre cursus scolaire/professionnel.