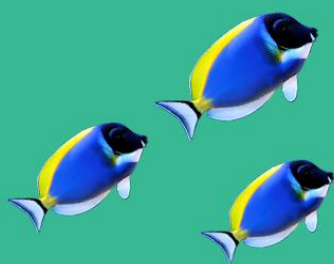




Bienvenue dans Snorkunking



PERRY Antoine
STREIFF Fanny



RAPPORT JAVA

Snorkunking

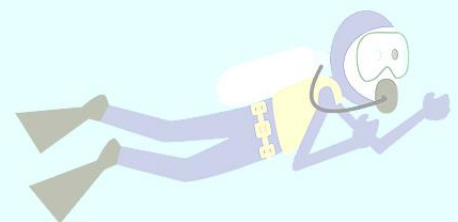
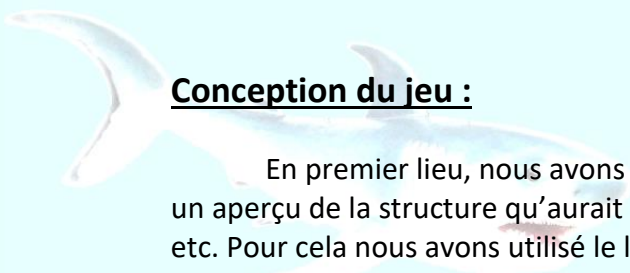
Ce jeu consiste en une plongée dans les eaux sous-marines à la recherche de trésors. Tel un pirate des temps modernes vous devrez parcourir les caves à la recherche de votre butin. Soyez le plus rapide dans votre tenue de plongée car vous devrez partager votre barre d'oxygène avec les autres joueurs ! Compétitif et haletant, bienvenue dans Snorkunking !

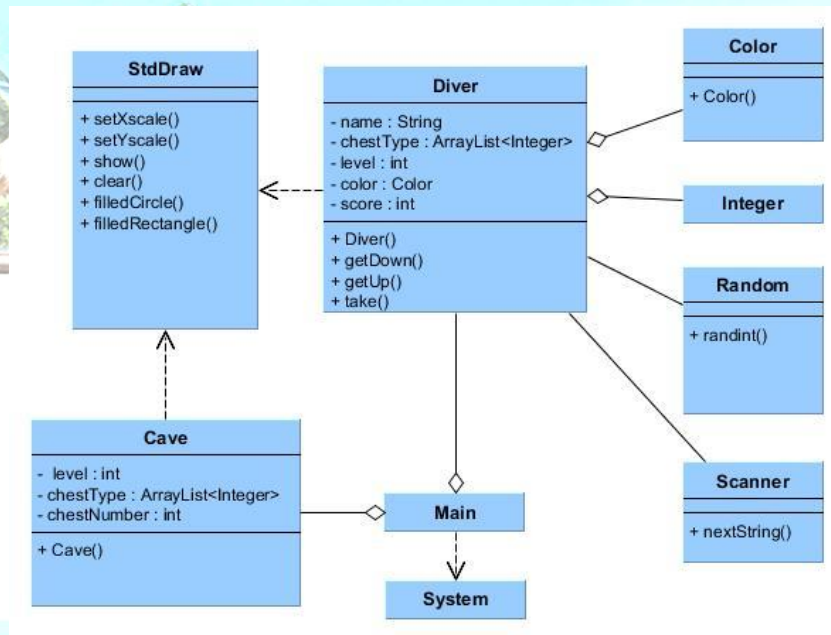
Règle du jeu :

Chaque partie est divisée en 3 manches, à chaque manche les joueurs plongent à la recherche de trésors. Les eaux sous-marines contiennent 3 caves toute plus profonde les unes que les autres. Chaque cave a un certain nombre de niveaux qui contient un coffre. Plus la cave est profonde, plus le coffre a de chance de contenir un trésor d'une plus grande valeur, jamais vous n'aurez autant envie de vous enfoncer en plein cœur des abysses. Une manche se finit lorsque la barre d'oxygène est vidée, si un joueur n'est pas remonté à temps il perd les coffres qu'il porte. Un coffre perdu tombe au fond de l'océan et il faudra aller dans la dernière cave pour le récupérer. Attention la barre d'oxygène se décharge facilement, chaque mouvement d'un plongeur consomme de l'oxygène et encore plus s'il porte un coffre !

Conception du jeu :

En premier lieu, nous avons essayé d'établir un premier diagramme UML afin d'avoir un aperçu de la structure qu'aurait notre jeu, des méthodes que nous devrions implémenter, etc. Pour cela nous avons utilisé le logiciel yEd afin de concevoir un diagramme en accord avec la représentation vue en cours.





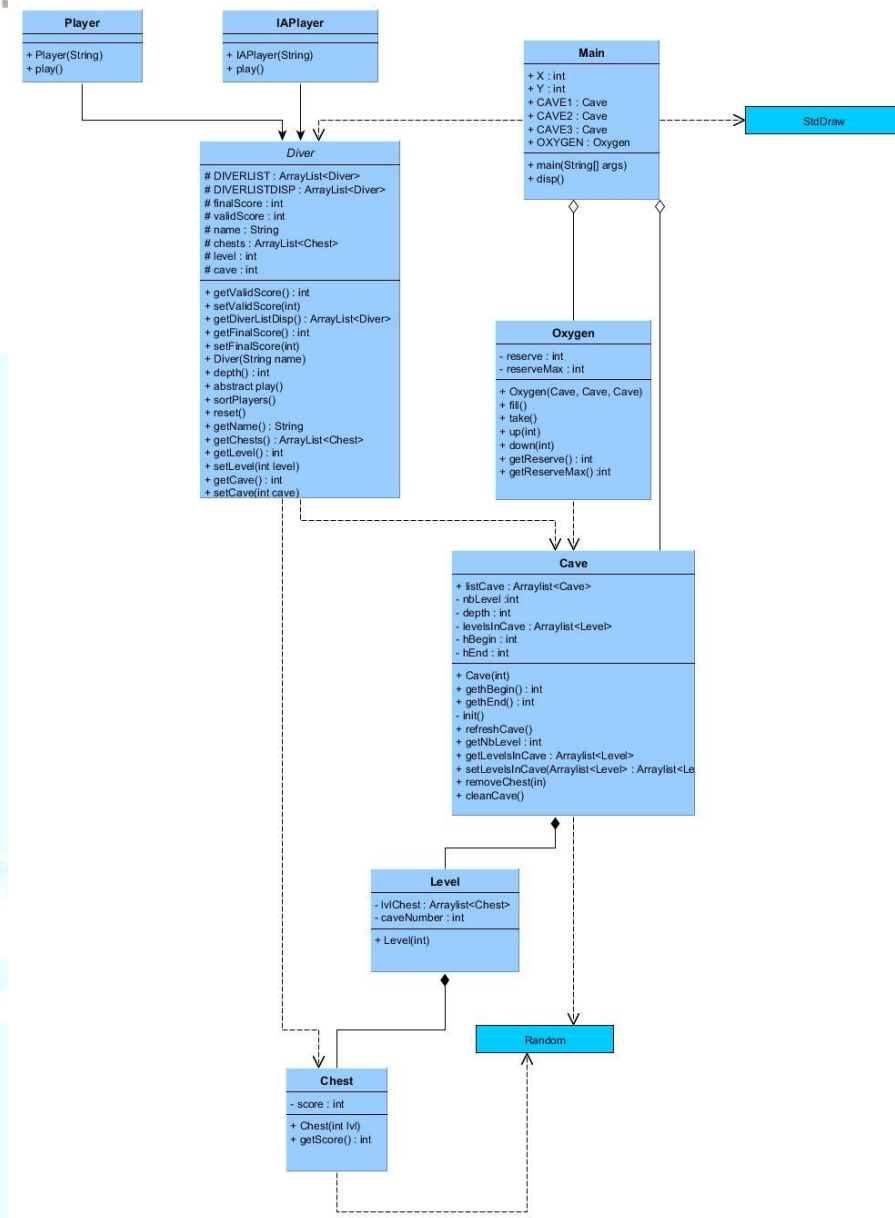
Après cette première représentation, nous avons commencé à implémenter le jeu en utilisant ce diagramme, mais en créant les classes, attributs, et méthodes, nous nous sommes rendu compte qu'il était, d'une part, incomplet mais également qu'il y aurait plein de problèmes auxquels nous n'avions pas pensés.

Pour faire face à ces défis, nous avons dû d'une part, revoir la conception de nos données, en augmentant le nombre de classes nécessaires, mais aussi bien cerner le fonctionnement de StdDraw, qui sera déterminant étant donné la dynamique du jeu : les niveaux peuvent disparaître, il ne s'agira donc pas d'afficher une barre d'oxygène, des joueurs et des coffres sur un simple background !

Une fois ceci fait, et après de nombreuses heures de codage et de réflexion effrénée notre jeu fonctionnait enfin ! Cependant nous ne savions pas ce qui allait nous attendre... (SPOILER : Beaucoup beaucoup beaucoup de bugs).

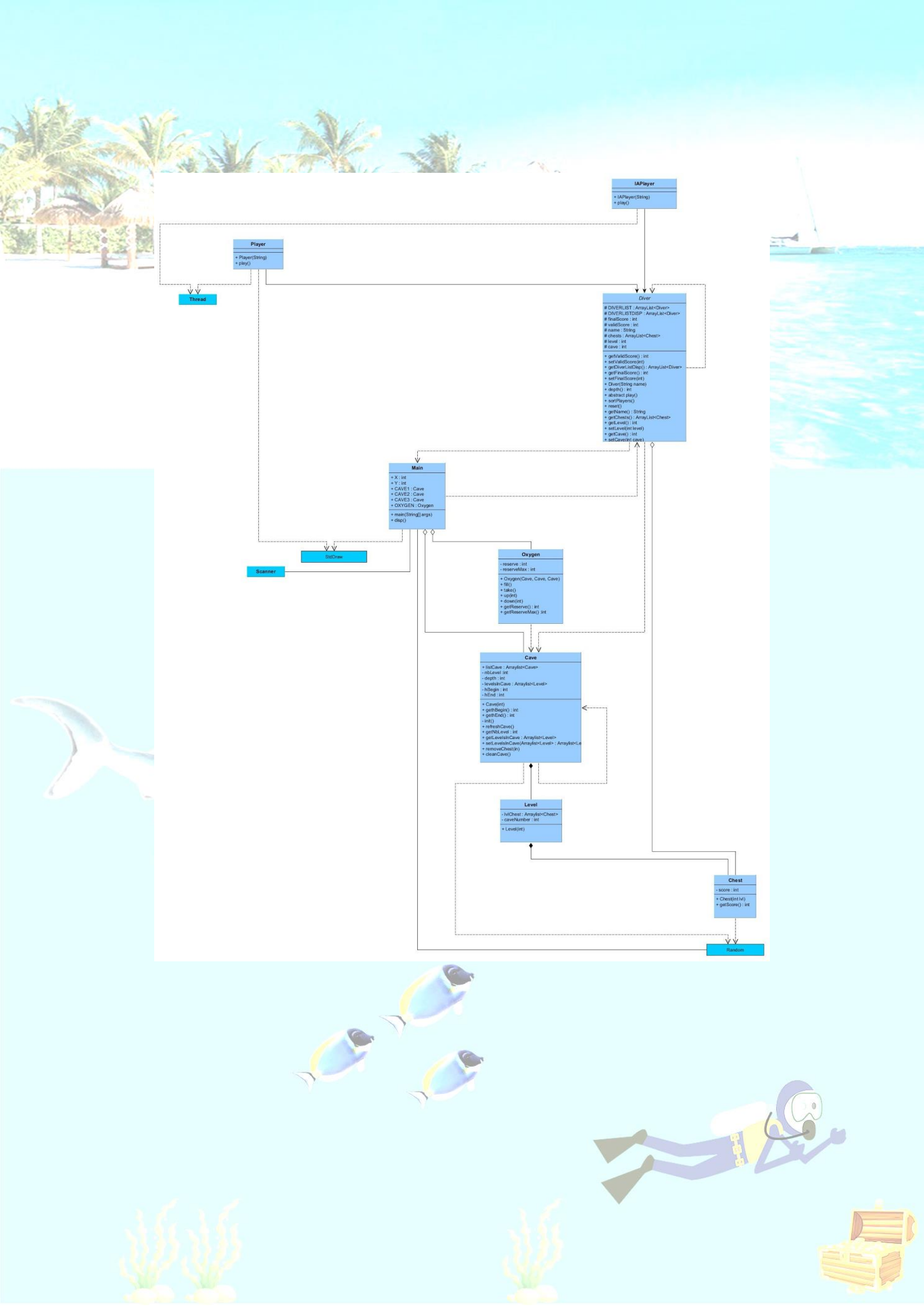
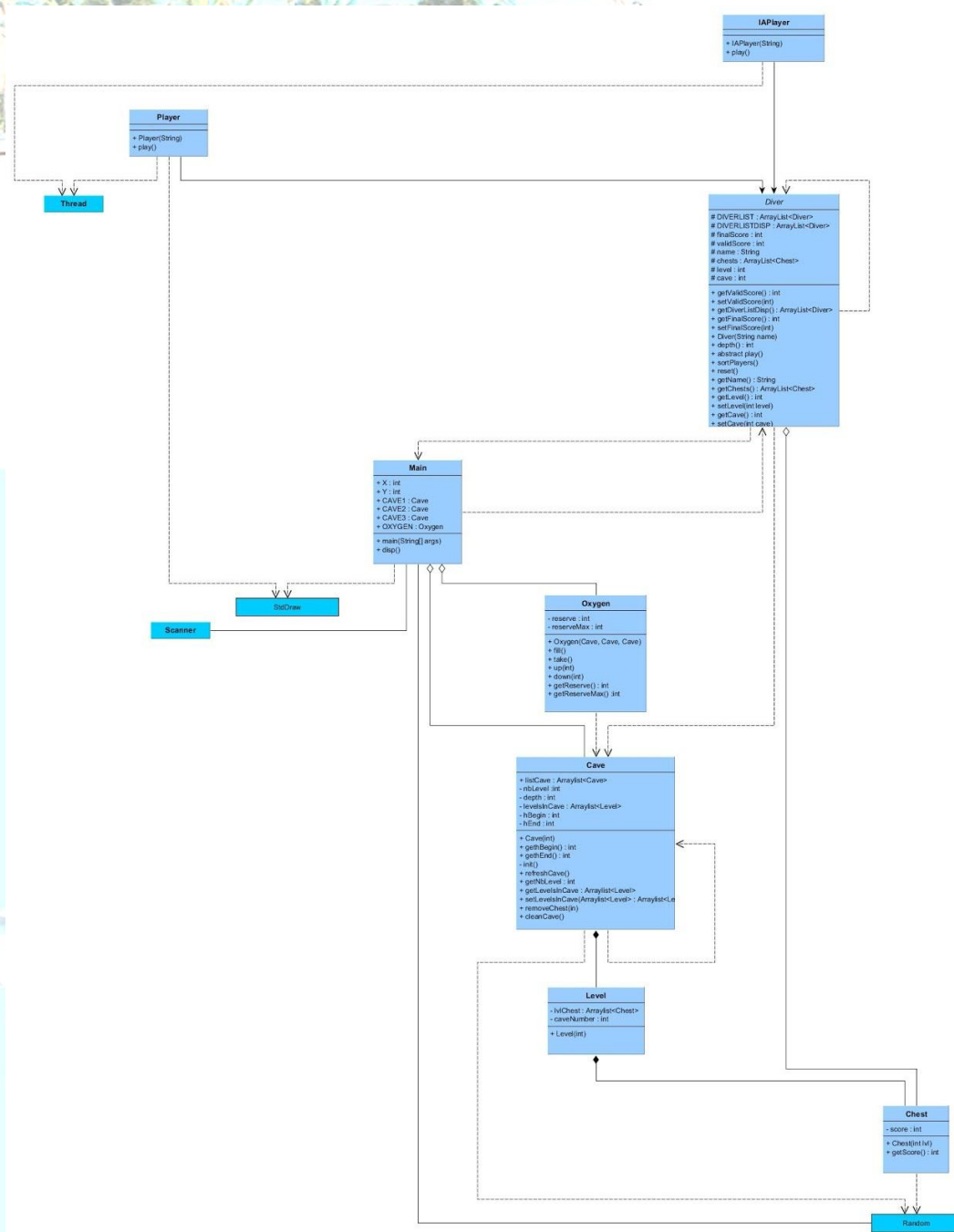


En effet, même si nous pouvions nous déplacer allègrement au fond des océans, récupérer des coffres, notre escapade sous-marine était joyeusement interrompue par des plongeurs découvrant le pouvoir de la téléportation, de barre d'oxygène se vidant à la vitesse de la lumière, de coffres qui une fois pris n'étaient pas comptabilisés, et même de certains `indexOutOfBoundsException` (quand nous prenions tous les coffres d'une cave par exemple).



C'est ainsi qu'est venue la décision de faire une version 2.0, qui corrigerait tous les bugs, tout en essayant d'optimiser un maximum nos méthodes, très longues et complexes pour la plupart.

Ainsi après de nombreuses heures de codage (encore une fois, la vie d'informaticien est dure), nous sommes parvenus au résultat final, sans bug et avec des méthodes beaucoup plus optimisées !



Analyse des éléments importants du projet :

1. IA

Pour cette partie, nous avons tiré profit du fait que nous utilisions une classe abstraite Diver qui était la super-classe de IAPlayer (ordinateur) et Player (joueur humain). Du coup, étant donné que les classes relatives au déplacement et à la prise de coffre étaient déjà créées, il ne manquait plus qu'à implémenter la "logique" de l'IA, c'est à dire dans quelles circonstances elle devait réaliser les actions.

Ainsi, nous avons opté pour une implémentation assez simple, celle de faire des allers-retours pour chaque coffre :

- Si on a un coffre on le remonte à la surface
- Sinon, si on a un coffre à notre niveau, on le prend
- Sinon, on descend

Cette technique n'a nécessité que quelques lignes et a même permis de battre un joueur humain, néanmoins elle montre vite ses limites et aurait pu être améliorée en une IA plus performante qui détermine si jamais elle doit monter ou descendre en fonction du nombre de coffres que chaque joueur a pris, en essayant de déterminer les mouvements qui viendraient à se produire.

2. Affichage des caves, niveaux et coffres

L'affichage dynamique n'a pas été chose aisée pour ce projet, en effet pour afficher un rectangle, on doit préciser les coordonnées du centre de celui-ci, et non de son contour, données que nous ne possédons pas forcément durant le jeu. Ainsi nous avons ajouté à chaque cave 2 attributs précisant l'ordonnée du haut et du bas de cette cave, on peut donc calculer la coordonnée du centre !

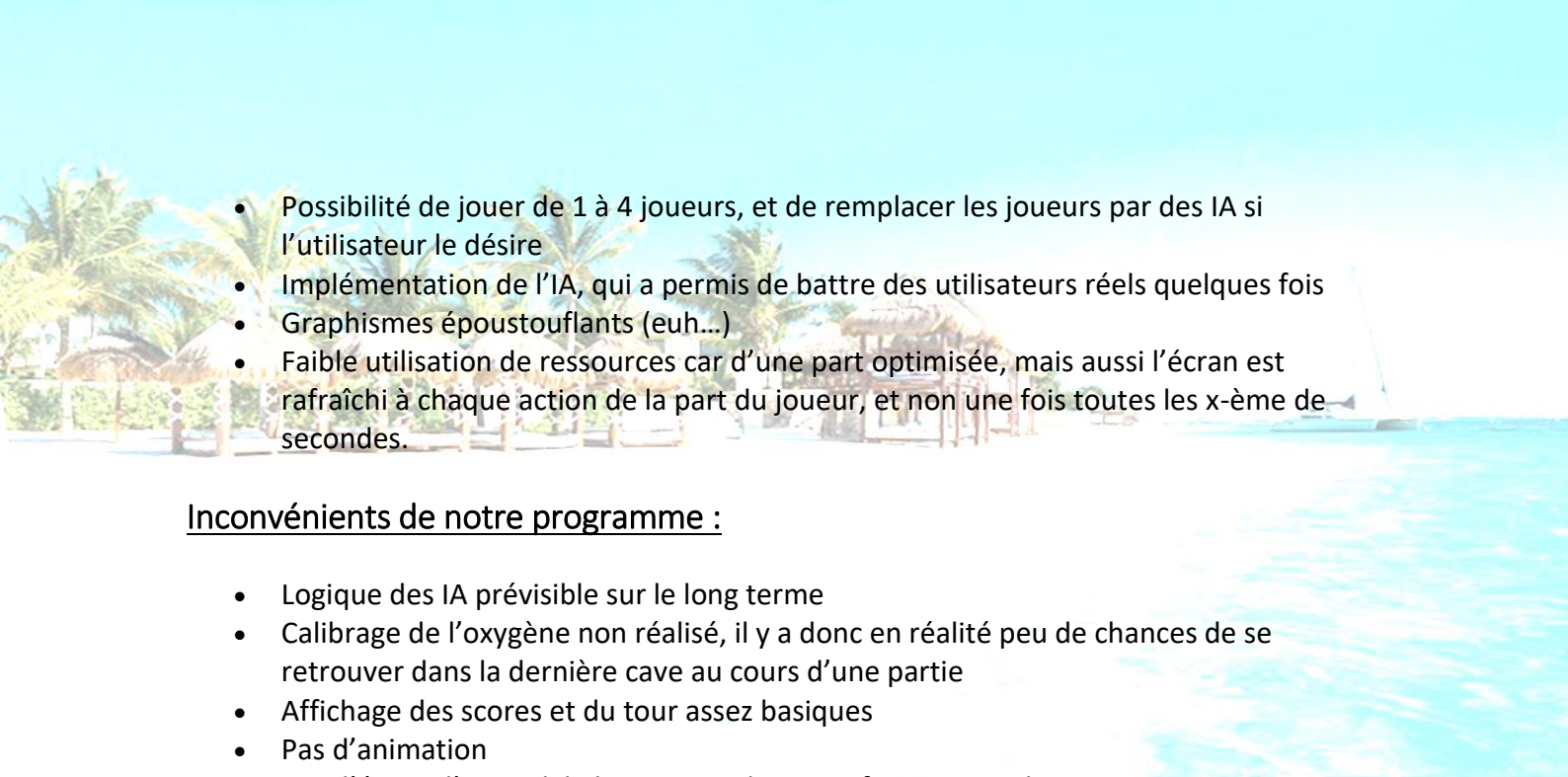
Ceci n'était qu'un exemple parmi d'autres : écroulement des caves, déplacement des joueurs entre les caves, etc, nous ont apporté quelques heures de chauffage de méninges !

3. Barre d'oxygène

Au début, l'affichage de l'oxygène restant était écrit de façon numérique. A la fin de la version 2.0, nous avons voulu ajouter la fameuse barre d'oxygène commune aux joueurs (c'est bien connu, des plongeurs respirent tous dans la même bouteille !), nous avons envisagé cela sous la forme d'un rectangle immobile, d'une couleur A, en fond, avec au-dessus le même rectangle, de couleur B, qui va, à mesure que l'oxygène se vide, s'écraser sur la droite, ainsi on a à l'écran une barre qui se vide !

Avantages de notre programme :

- Structure de données représentant bien le concept réel du jeu : classe Cave qui contient des objets de la classe Level qui eux contiennent des objets de la classe coffre, coffre qui sont ouvert en surface mais qui contiennent toujours le même nombre de trésors, au lieu d'être calculé randomly à la sortie de l'eau

- 
- Possibilité de jouer de 1 à 4 joueurs, et de remplacer les joueurs par des IA si l'utilisateur le désire
 - Implémentation de l'IA, qui a permis de battre des utilisateurs réels quelques fois
 - Graphismes époustouflants (euh...)
 - Faible utilisation de ressources car d'une part optimisée, mais aussi l'écran est rafraîchi à chaque action de la part du joueur, et non une fois toutes les x-ème de secondes.

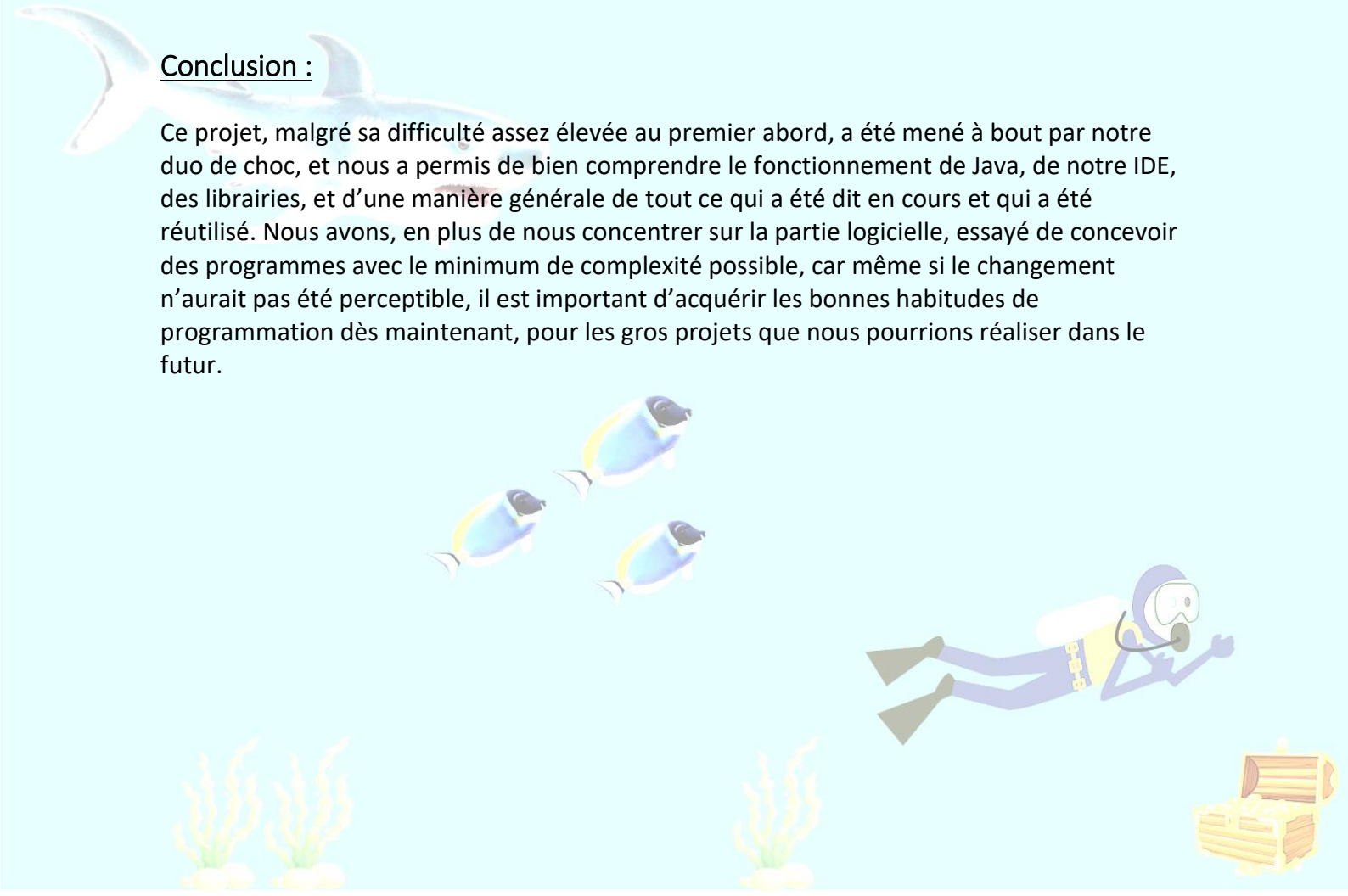
Inconvénients de notre programme :

- Logique des IA prévisible sur le long terme
- Calibrage de l'oxygène non réalisé, il y a donc en réalité peu de chances de se retrouver dans la dernière cave au cours d'une partie
- Affichage des scores et du tour assez basiques
- Pas d'animation
- Pas d'écran d'accueil, le lancement du jeu se fait en console
- Utilisation de StdDraw, librairie assez limitée en termes de possibilités graphiques

Ce que ce jeu nous a apporté :

C'était pour nous une nouvelle expérience de s'attaquer à un jeu de cette envergure, cela nous a appris à mettre en route un projet, à le maintenir et mettre à jour en utilisant de nouveaux outils (Git, yEd, etc), et à approfondir nos connaissances en Java, car les notions appelées tout au long de la programmation ont été diverses et variées.

Conclusion :



Ce projet, malgré sa difficulté assez élevée au premier abord, a été mené à bout par notre duo de choc, et nous a permis de bien comprendre le fonctionnement de Java, de notre IDE, des librairies, et d'une manière générale de tout ce qui a été dit en cours et qui a été réutilisé. Nous avons, en plus de nous concentrer sur la partie logicielle, essayé de concevoir des programmes avec le minimum de complexité possible, car même si le changement n'aurait pas été perceptible, il est important d'acquérir les bonnes habitudes de programmation dès maintenant, pour les gros projets que nous pourrions réaliser dans le futur.