Projet d'informatique

ECE Arena

Table des matières

Présentation générale

Ce semestre, vous allez pouvoir développer un jeu de combat au tour par tour, inspiré du jeu **Dofus Arena**. L'idée générale est d'organiser des combats entre joueurs.

Vous utiliserez pour cela la bibliothèque graphique Allegro (4 à Paris, 5 à Lyon) afin de réaliser votre jeu en 2D.

Le jeu original se joue normalement en réseau. N'ayant pas encore suivi de cours de réseaux, vous développerez ce jeu de telle sorte que tous les joueurs jouent sur le même ordinateur, l'un après l'autre.

Vous réaliserez ce projet en équipe *(les détails de la constitution des équipes vous est donnée sur la page Moodle de votre campus)*, mais la note pourra être individualisée. Au sein d'une même équipe de projet, chaque étudiant.e devra s'investir à parts égales dans la conception et le développement.

Vous pouvez voir un exemple de gameplay du jeu original dans la vidéo suivante :
https://www.youtube.com/watch?v=27AJExdmMsY

Consignes

- L'étape de conception est primordiale. Avant de commencer à coder, vous devez avoir identifié chaque fonctionnalité, réfléchi à la manière de les implémenter (algorithmes, stockage des données…), vous devez anticiper leur assemblage. Cette étape de conception vous permettra d'anticiper chaque problème qui vous obligerait à recommencer tout ou une partie de votre code. Une bonne conception permet d'aller beaucoup plus loin, avec moins de soucis.
- Vous avez le choix du thème de votre jeu. Vous pouvez par exemple réaliser votre jeu sur le thème d'un jeu existant, d'un film, d'un roman, etc.
- Vous devez mettre en place un *versionnage* de votre code avec **Git** (informations sur Moodle), et effectuer des *commits* réguliers, dès le début. Ce *versionnage* a quatre intérêts :
  - Pouvoir revenir à une version antérieure du code à tout moment,
  - Eviter de perdre du code en cas de soucis,
  - En cas de plagiat, vous pourrez prouver l'appartenance du code,
  - Le jury pourra vérifier l'implication de chacun.e.
- **TOUT PLAGIAT EST STRICTEMENT INTERDIT :** Conformément au règlement intérieur de l'école que vous avez lu et signé, il est strictement interdit d'utiliser le code d'un autre groupe, d'une tierce personne, ou que vous avez trouvé sur le web sans citer la source et sans l'accord de votre enseignant.e. Les pénalités liées au plagiat sont lourdes et pourront entraîner des répercussions au-delà du cours lui-même (exclusion de l'école).

Glossaire des acronymes

PA	Points d'Action
PM	Points de Mouvement
PNJ	Personnage Non Jouable
PV	Points de Vie

Marche à suivre

Comme vous avez du vous en rendre compte lors de votre premier projet d'informatique, la conception est primordiale. Bien que vous ayez envie de commencer à coder le plus rapidement possible, nous vous le déconseillons fortement, d'autant plus pour ce projet qui nécessite de réfléchir à certains algorithmes et d'avoir une vision globale du déroulement d'une partie. Réaliser une bonne conception vous ralentira un peu au début, mais vous permettra d'aller beaucoup plus loin et vite, avec beaucoup moins de soucis et d'imprévus.

Avant de coder, vous devez savoir précisément à quoi ressemblera votre jeu (thème, visuels), quels seront les différents affichages (interfaces identifiées avec la méthode DTI) et comment les joueurs passeront d'un affichage à un autre (cheminements). Avec Allegro, vous devez penser aux éventuelles animations (comment animer des personnages, lors de leur déplacement ou d'attaques par exemple). Vous devez également décider comment vous allez stocker les données identifiées lors de votre analyse DTI : dans des structures, tableaux, tableaux à deux dimensions (matrices), files, piles… ? Comment éviter d'avoir des doublons de données grâce aux pointeurs. Vous devez anticiper la façon dont toutes ces données seront transformées en une interface à l'écran.

Nous vous conseillons pour cela de lire et relire minutieusement le cahier des charges en surlignant les passages importants, et en utilisant les méthodes DTI et ACD, avant de commencer à coder. Faites cette étape seul.e, puis avec votre équipe, sous la forme de *brainstorming* (autour d'un tableau blanc par exemple). A l'issue de la conception, tout le monde doit avoir une vision précise de la forme que prendront le jeu et le code.

Une fois les différentes sous-tâches identifiées en ACD, vous pourrez mieux vous les répartir entre vous, et éviter de recoder plusieurs fois une même fonctionnalité.

Nous vous conseillons également de désigner un.e chef de projet, qui répartira les tâches et veillera au bon avancement du projet. Vous pouvez pour cela utiliser différents outils de gestion de projets : Monday, **Trello**… *(créez trois colonnes : "A faire", "En cours", "Fait". Ajoutez les tâches sous forme de cartes et assignez des personnes à chaque tâche, avec une deadline)*.

Au niveau de l'implémentation, nous vous déconseillons fortement de mélanger le code de traitement des données (actualisation des positions…) et le code qui gère l'affichage (fonctions de dessin). Tout l'affichage doit se faire dans une même procédure, elle-même décomposée en plusieurs sous-procédures. Lorsqu'un personnage avance par exemple, nous modifions dans un premier temps ses données (sa position) et éventuellement celles des autres éléments du jeu, puis, dans un second temps, quand toutes les données du jeu auront été mises à jour, nous procéderons au réaffichage intégral de l'image en utilisant les données actualisées.

Le jeu comportera beaucoup d'images. Vous devez réfléchir à leur stockage. Si deux éléments ont la même apparence (image), il est inutile de charger cette image deux fois. En revanche, rien n'empêche de charger l'image une fois, et de stocker son adresse mémoire à plusieurs endroits. Nous vous conseillons d'ailleurs de commencer à réaliser votre jeu sans image, représentez par exemple les personnages par des cercles. Vous pourrez aisément remplacer le cercle par une image lors que tout fonctionnera si vous avez bien décomposé votre code.

Enfin, **TESTEZ** au fur et à mesure, et travaillez sur le même projet. Si vous travaillez sur des projets différents, il vous sera difficile de les fusionner. Commencez à coder la base de votre jeu sur un seul ordinateur, puis, lorsqu'elle est assez conséquente, répartissez-vous les tâches sur plusieurs

ordinateurs. Si vous codez à plusieurs sur un petit projet, vous risquez d'avoir beaucoup de conflits dans Git ! Il vaut mieux d'abord faire grandir la base.

Cahier Des Charges

Objectif

L'objectif de ce jeu de combat au tour par tour est de remporter le plus de combats en arène afin de gagner en expérience. L'expérience permettant de gagner des niveaux qui permettent de gagner des points de vie (PV), des points d'action (PA), ou encore des sorts supplémentaires.

Afin d'alléger le jeu, nous ne développerons que la partie "combat". Les joueurs ne gagneront pas d'expérience. Ils auront tous le même nombre de PV, de PA, de PM et de sorts. Ils auront en revanche le choix parmi 4 "Classes" (~personnages) ayant chacune des caractéristiques différentes (apparence et sorts).

Un classement sera affiché à la fin de chaque combat.

Matériel

Le jeu se jouera entièrement à la souris (par clics).

Le clavier pourra être utilisé en complément de la souris. Mais le jeu devra être fonctionnel sans clavier. Il pourra par exemple être utilisé pour saisir le nombre de joueurs (en plus de la souris donc), afficher un menu permettant de quitter ou commencer une nouvelle partie. Il pourra également être utilisé pour saisir le nom des personnages (facultatif) ou effectuer des raccourcis (taper 1 pour activer le sorts n°1…).

Classes (personnages)

Au début du jeu, chaque joueur à le choix entre plusieurs Classes (de personnages). La classe d'un personnage définit son apparence ainsi que les sorts dont il dispose.

Les Classes du jeu Dofus vous sont présentées sur la page suivante :
https://www.dofus.com/fr/mmorpg/encyclopedie/classes.

Vous y trouverez pour chaque Classe la liste des sorts qu'elle octroie. Vous pouvez vous en inspirer.

Interfaces

Notre jeu se décomposera en 4 interfaces (étapes) :

Au lancement du jeu, un menu demande le nombre de joueurs (de 2 à 4). Il est à choisir à la souris, en cliquant sur le bouton correspondant au nombre de joueurs souhaité. Le clavier peut être utilisé en complément.

Une fois le nombre de joueurs défini, chaque joueur choisit l'un après l'autre la Classe de son personnage parmi une liste prédéfinie (que vous pouvez inventer d'après votre thème).

Enfin, la partie commence : les joueurs sont placés aléatoirement sur la grille *(bonus : chaque joueur décide en même temps où placer son personnage dans un laps de temps défini)*, puis chaque joueur

joue à tour de rôle (ordre fixé aléatoirement au début de la partie, mais permanent ensuite), jusqu'à ce qu'il ne reste qu'un.e seul.e survivant.e.

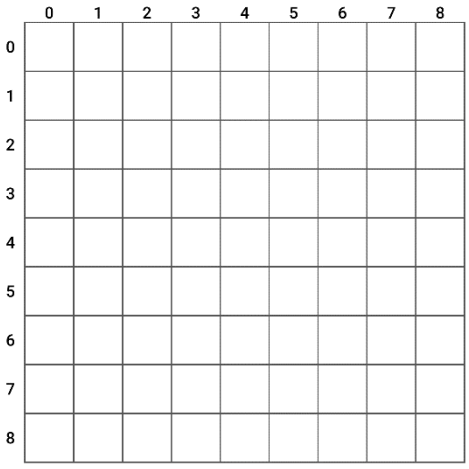
A la fin de la partie, le classement est affiché, puis trois boutons permettent de :

- Quitter,
- Rejouer la même partie (revanche avec les mêmes joueurs/personnages),
- Lancer une nouvelle partie (retour au choix du nombre de joueurs).

L'interface d'une partie de jeu devra faire figurer :

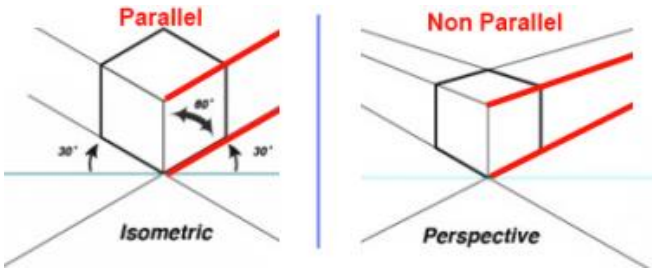
- Le décor (paysage, avec obstacles (buissons, lacs, troncs d'arbres…))
- Les personnages
- La liste des personnages, afin de savoir à qui c'est le tour, et qui est le suivant
- Les PV de chaque personnage
- Pour le personnage dont c'est le tour :
    - Un bouton pour attaquer au corps à corps
    - Un bouton par sorts
    - Le nombre de points de mouvements
    - Le nombre de points de vie
    - Le nombre de points d'action
    - Un bouton "Joueur suivant", que le joueur peut utiliser quand il a fini ses actions

Conception et affichage de l'arène



Afin de simplifier les déplacements, nous travaillerons sur des arènes suivant un repère orthogonal, comme c'est le cas dans le jeu Pokemon sur Gameboy :

Si vous le souhaitez, vous pouvez créer vos cartes de façon isométriques (fausse perspective), comme c'est le cas dans le jeu Dofus original, cela implique quelques calculs supplémentaires. Réfléchissez à tout ce que cela implique lors de votre phase de conception :



Pour concevoir vos arènes, vous pouvez utiliser n'importe quel logiciel de dessin, veillez juste à respecter la position des cases de votre grille de jeu. Certains logiciels tels que Tiled (https://www.mapeditor.org/) sont spécialisés dans l'élaboration de ce genre de cartes et pourront vous être d'une grande aide.

NB : Bien que cette tâche graphique puisse prendre beaucoup de temps, il est exclu qu'une personne ne fasse que ça. Chaque membre de l'équipe doit coder à parts égales. De plus, évitez de perdre trop de temps sur la création des images.

Chargement de l'arène

Une seule arène est demandée. Mais si vous concevez bien votre jeu, il sera possible d'en charger d'autres facilement et de lancer chaque nouvelle partie dans une arène aléatoire (ou de la choisir).

Pour charger une arène, plusieurs choix s'offrent à vous : vous pouvez charger une image (.jpg par exemple) et coder en dur dans votre code la position des éléments du décors (murs, troncs, lacs, etc.), mais ce n'est pas très réutilisable…

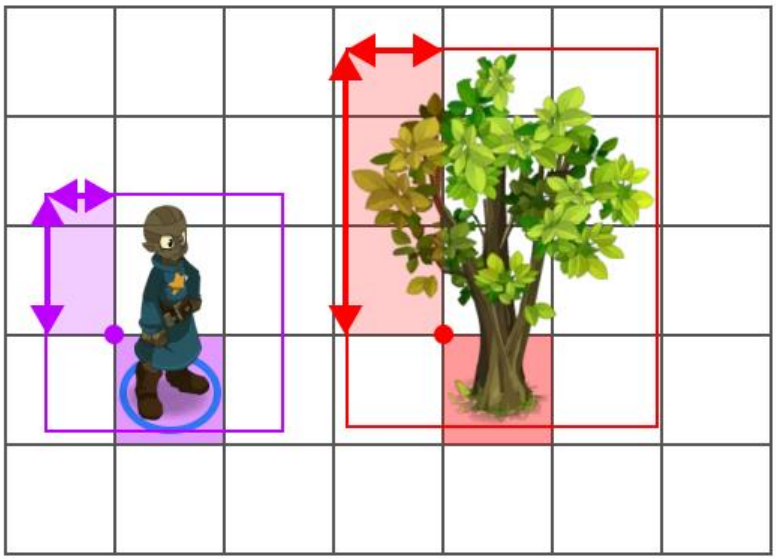
Une meilleure solution serait de charger l'image de l'arène et de stocker ses informations dans un fichier texte du même nom (.txt) contenant une matrice d'entiers. Un 0 pourrait représenter une case sur laquelle on ne peut pas marcher, un 1 une case sur laquelle on peut marcher.

BONUS : Pour aller plus loin, l'image de l'arène pourrait être générée à partir d'un ou plusieurs fichier(s) texte (exemple : 2 = herbe, 3 = sable, 4 = rocher, 5 = mur…). L'avantage serait de pouvoir créer des arènes très facilement en créant un fichier texte.

Nous vous conseillons de commencer sans image, avec des formes simples (rectangles, cercles…), en dessinant la grille (succession de carrés) avec des codes couleurs, sans aucun obstacle, puis de rajouter progressivement les obstacles, puis les images, etc. Passer d'un triangle à une image ne demande la modification que d'une seule ligne de code si vous n'avez pas de duplication de code !

Pour un affichage plus réaliste, il faut parfois sortir des cases…

Pour donner une fausse impression de 3D (de profondeur), vous pouvez faire dépasser les éléments de leur case :



Comme nous dessinons les images en donnant leur coordonnée (x1,y1) correspondant au coin en haut à gauche, il faut alors prendre en compte le décalage entre la case où se trouve l'élément à afficher, et l'endroit où doit se trouver son coin supérieur gauche.

L'image de l'arbre est disponible ici : https://dofuswiki.fandom.com/wiki/Ash

De plus, pour que les personnages ne s'affichent pas par-dessus les arbres, il faut d'abord dessiner le fond, puis les personnages, puis les arbres. Si aucun de vos éléments ne dépasse de sa case, vous n'avez pas toutes ces problématiques, mais le rendu est moins qualitatif.

Rappel : TESTEZ au fur et à mesure que vous codez afin de pouvoir identifier plus facilement les nouvelles lignes de code qui pourraient causer un ralentissement, provoquer un bug....

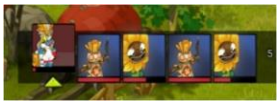
Tour

Lorsque c'est à un joueur de jouer, il dispose de 15 secondes (durée de votre choix) pour faire ses actions, dans n'importe quel ordre :

- Se déplacer, dans la limite de ses PM,
- Attaquer, dans la limite de ses PA,
- Passer son tour.

Au bout de 15 secondes, le jeu passe automatiquement au personnage suivant. Si le joueur termine ses actions avant les 15s, il peut cliquer sur le bouton "Joueur suivant".

Dans l'interface, l'ordre des joueurs ainsi que le joueur actif sont mis en évidence :

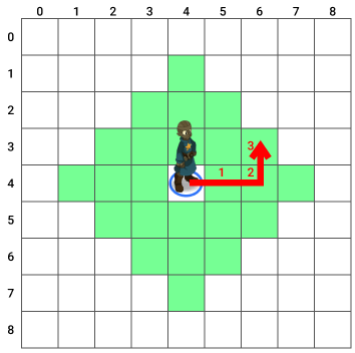


Déplacements

Dans cette partie, nous allons parler de coordonnées. Veillez bien à ce que tous les membres de votre équipe utilisent le même langage : il peut y avoir de grandes ambigüités entre les différentes notations. Notamment entre la notation mathématique (x, y) et la notation des tableaux en C : tab[ligne][colonne] qui reviendrait à écrire : tab[y][x] et non pas tab[x][y]. Dans votre conception, privilégiez l'emploi des termes "ligne" et "colonne", plutôt que y et x.

Les personnages se déplacent dans l'arène selon une grille (déplacement horizontal et/ou vertical uniquement). Il est impossible de stagner entre deux cases, et le déplacement se fait uniquement au clic de la souris (sans dépasser le nombre de Points de Mouvement disponibles qui est de 3PM / tour / personnage).

Dans le schéma ci-dessous, les cases vertes représentent les cases sur lesquelles le personnage peut se déplacer avec 3PM. Notez qu'il n'y a ici aucun obstacle.



Le joueur est en [4][4]. Si chaque case fait 100px à l'écran (supposition), et que l'on clique en (604, 339), le personnage devra se déplacer (s'il le peut) en case [3][6]. Il suffit de diviser la coordonnée du clic par la taille de chaque case pour obtenir l'indice de la ligne/colonne sur la grille (dans le tableau).

Comme le personnage se trouve en [4][4], il devra d'abord aller en [4][5], puis en [4][6], puis en [3][6]. Il peut aussi aller en [3][4], puis en [3][5], puis en [3][6].

Pour déplacer un personnage à la souris, il suffit au joueur de survoler la carte avec la souris. Si le déplacement vers la case survolée est possible, le chemin apparait alors en vert comme dans l'image ci-dessous où le joueur dispose visiblement de 4PM. *Evitez de rechercher un chemin à chaque fois que le joueur bouge la souris d'un pixel, essayez plutôt de détecter le changement de case, qui déclenchera ensuite la recherche de chemin.*

Vous pouvez stocker le chemin dans un tableau de structures Coords contenant la colonne et la ligne : [(2,3)][(2,4)][(3,4)]. Ainsi, avec des boucles, le personnage glissera progressivement vers la case (2,3) (donc (200px,300px)), puis sur l'axe y jusqu'à atteindre la case (2,4), puis enfin sur l'axe x jusqu'à ce que son x vaille 300.

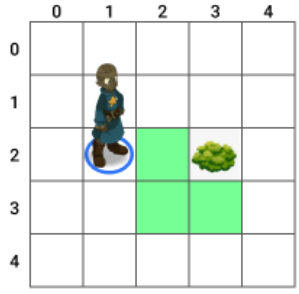
Vous noterez que dans le tableau **[(2,3)][(2,4)]**[(3,4)], la somme des valeurs absolues de la différence sur l'axe x et sur l'axe y est à chaque fois égale à 1. Car le personnage se déplace de case en case et donc de 1 en 1 : |2-2| + |3-4| = 1.

Pour construire ce tableau de coordonnées. Vous pouvez prioriser le déplacement sur l'axe x, puis si ce déplacement n'est pas possible, sur l'axe y.

Ainsi, si nous sommes en (3,5) et souhaitons aller en (5,4), il va falloir aller deux fois à droite. Le x de destination est plus grand donc on va incrémenter de 1 le x dans chaque case s'il n'y a pas d'obstacle en (4,5), on peut ajouter cette coordonnée au tableau. Puis recommencer, nous arrivons en (5,4). Nous avons atteint le bon x. Au tour du y. Il est plus petit cette fois ci, nous allons donc monter en (5,4). La taille logique de notre tableau est maintenant de 3, nous n'aurions pas pu faire plus, heureusement, nous sommes arrivés à notre destination, le déplacement est possible.

Pour animer ce déplacement, si nous sommes en (3,5) (en [5][3] ?), cela veut dire que notre personnage est en (300,500) si chaque case fait 100px. Nous allons donc déplacer notre personnage 2px par 2px vers sa prochaine case : (4,5), soit (400,500). Le personnage va donc faire (300,500), (302,500), (304,500), … jusqu'à (400,500).

En cas d'obstacle (élément de décor ou autre joueur) ?



Dans la situation ci-contre, un buisson fait obstacle. Nous sommes en (1,2) (en [2][1]), si le joueur place la souris en (3,3), il va falloir trouver un chemin vers cette case ! Si l'on suit la procédure, nous commençons d'abord par vérifier si le déplacement est possible sur l'axe x (dans le cas où notre x est différent de celui de destination) : nous sommes en 1 et souhaitons aller en 3, il y a une différence de 2, il va donc falloir aller à droite deux fois, essayons d'une case, en (2,2), c'est possible. Ensuite, même procédure : nous sommes en (2,2) et souhaitons aller en (3,3), il faut donc ajouter 1 à x. Testons la case (3,2) : non. Il y a un buisson. Nous allons donc essayer de nous rapprocher de notre destination via l'axe y, le y de destination est supérieur à notre y, nous allons donc tester la case y+1 (2,3) : c'est bon. On recommence, on essaye de se rapprocher sur l'axe x puisqu'il y a une différence entre notre x et celui de destination, et ça tombe bien car la case voisine est celle de destination. Le chemin est donc possible avec 3PM. Il faut juste stocker ces cases dans un tableau de coordonnées pour mémoriser le chemin à emprunter.

Si on avait survolé la case (4,3), 3PM n'auraient pas suffi. Il faut que l'algorithme s'arrête à 3 déplacements.

Si votre groupe ne se sent pas capable de gérer les déplacements, vous pouvez faire un avancement case par case.

Attaques

Le but du jeu étant de gagner la bataille, chaque joueur, peut, lorsque c'est son tour, lancer des sorts ou attaquer au corps à corps.

Lorsqu'un personnage subit une attaque, il perd des PV calculés aléatoirement en fonction de la puissance de l'attaque. Par exemple une attaque enlève 8PV, plus ou moins 4PV, et elle a une chance sur 12 d'échouer.

Attaque au corps à corps 

L'attaque au corps à corps n'est possible que sur les cases adjacentes (voisines).

Elles sont beaucoup moins puissantes qu'un sort, mais ne coûtent que 2PA. Elles peuvent enlever entre 1 et 5PV à l'ennemi, avec 1 chance sur 10 d'échouer (et donc n'enlever aucun PV). Vous pouvez personnaliser toutes ces valeurs.

 Attaque par sort

Chaque Classe de personnage offre aux joueurs différents sorts. A vous de les inventer. Vous pouvez vous inspirer des sorts du jeu original. Dans notre version ECE Arena, chaque Classe aura 4 sorts. Soyez créatifs, vous pouvez intervenir sur leurs caractéristiques qui sont les suivantes :

- Chaque sort a une portée minimale, ainsi un sort peut ne pas être lancé sur un ennemi se trouvant sur la case voisine, car il est trop proche !
- Chaque sort a une portée maximale.
- Chaque sort coute des PA, plus il est puissant, plus il coute de PA.
- Chaque sort enlève entre *val1* et *val2* PV à l'ennemi, et a *val3*% de chance d'échouer. Les termes *val1*, *val2*, et *val3* sont donnés à titre indicatif et ne représentent en aucun cas de bons noms de variables.
- Bonus : un sort peut toucher une zone de plusieurs cases (la case visée + les 4 autour).
- Bonus : un sort peut aider : en octroyant des PV, PM ou PA au joueur lui-même, ou à un.e allié.e dans le cas où vous gérez les équipes (autre bonus).
- Bonus : inventez d'autres sorts ayant leurs propres caractéristiques !

Comment jeter un sort ?

Lorsque c'est à son tour, un joueur peut : cliquer sur une case pour s'y déplacer (si le déplacement est autorisé, nombre de PM suffisant), ou attaquer. Pour attaquer, le joueur devra cliquer sur le bouton de l'attaque (image à gauche). Chaque case dans la portée de l'attaque se mettra alors en surbrillance comme dans l'image ci-dessous :

Portée des sorts

Voici des exemples de portées de sorts :



Pour savoir si une case est à la portée d'un sort, il faut que la somme des valeurs absolues de la différence sur l'axe x et sur l'axe y soit entre la portée minimale et la portée maximale : par exemple, si notre joueur est en (5,6), pour savoir quelles cases mettre en surbrillance, nous parcourons toute la grille du plateau et pour chaque case (x,y), nous la mettons en surbrillance si : portée min <= |5-x|+|6-y| <= portée max.

Pour simplifier le jeu, les sorts pourront être lancés sans prendre en compte les éventuels obstacles. La prise en compte des obstacles pour les sorts est en bonus.

Animations

Ce sont les animations qui rendront votre jeu qualitatif. Vous pouvez très bien les rajouter à la fin, mais anticipez leur ajout : qu'est-ce que cela impliquera ?

Deux éléments peuvent être animés :

- Les personnages,
  - Effet de "respiration", comme dans la vidéo Youtube présentée en tout début de ce sujet, où les personnages ne sont jamais statiques, même s'ils ne font rien,
  - Lors de leur déplacement, afin de donner l'impression que les personnages marchent réellement,
- Et les attaques.

Si vous souhaitez limiter les animations, pensez à un thème qui s'y prête : par exemple, des personnages en lévitation nécessiteront moins, voire aucune animation, alors que des personnages disposant de jambes nécessiteront une animation de celles-ci afin d'être crédibles.

Lors des attaques, vous pouvez ajouter des effets de sorts, d'explosion par exemple, à l'endroit où est lancée l'attaque.

Les animations sont probablement l'élément le plus difficile à intégrer au jeu puisqu'elles durent quelques millisecondes. Vous devez anticiper leur intégration et être capable de les stocker sous forme d'images successives (dans un tableau de bitmaps), en plus de savoir quand les afficher (à quelle étape).

Méthodologie

Identifiez chaque fonctionnalité nécessaire (déplacer un personnage…), puis, pour chacune d'entre-elles, identifiez différents niveaux possibles :

- Déplacement, niveau 1 : on vérifie si le déplacement est possible, et si c'est le cas, on téléporte le personnage (aucune transition/animation),
- Déplacement, niveau 2 : idem que le niveau 1, mais on fait glisser le personnage de case en case jusqu'à sa destination,
- Déplacement, niveau 3 : idem que le niveau 2, mais les jambes sont en plus animées.

Si vous décomposez bien votre code en plusieurs fonctions, il vous sera possible de passer d'un niveau à l'autre en ne modifiant qu'une seule fonction. Pour cela, évitez d'éparpiller votre code et ne mélangez pas les fonctions : si une fonction s'appelle "dessinerPersonnage", elle ne doit faire que ça, elle ne doit en aucun cas effectuer de déplacement du personnage (modification de ses coordonnées) ou quoi que ce soit.

Bonus

Vous devez inventer et implémenter une ou plusieurs extensions au cahier des charges. Cela peut être un mode de jeu contre une Intelligence Artificielle, ou encore l'ajout d'une fonctionnalité. Soyez créatifs !

En fonction de la qualité de votre/vos extension(s) et du défi technique qu'elle(s) représente(nt), un bonus pouvant pourra vous être attribué sur la note "Jeu" (voir Barème).

Exemples de bonus :

- Combat en équipes (chaque joueur apparaitra sur un "halo" de la couleur de son équipe (rouge ou bleu)),
- Génération de l'arène à partir de fichiers texte (matrice d'entiers où 1 = herbe, 2 = sable, 3 = pavés, 4 = eau, 5 = buisson…)
- Gain d'expérience, de niveaux et de sorts avec sauvegarde des joueurs à la fin de chaque partie,
- Chargement des Classes depuis un fichier,
- Ajout de PNJ (pilotés par une Intelligence Artificielle),
- Sorts spéciaux (sorts qui limitent le nombre de PA sur 2-3 tours, sorts qui attaquent une zone…),
- Invocation de PNJ dotés d'une Intelligence Artificielle,
- Affichage de l'arène de façon isométrique,
- Diffusion d'une musique d'ambiance,
- Etc.

Evaluation

Une note sur 20 sera attribuée à l'équipe. Elle sera inspirée du barème suivant (qui est indicatif) :

Jeu Sur 20 (+5 pts bonus)	Respect du cahier des charges	/10
	Qualité du code source	/5
	Jouabilité Le programme compile, et permet de jouer à une partie.	/3
	UI/UX Le jeu s'affiche bien, l'interface est réfléchie et facile à utiliser.	/2
	Extensions (bonus) Voir partie "Extensions"	+5 (bonus)
Organisation Sur 10	Organisation de l'équipe Méthodes mises en place par l'équipe pour s'organiser.	/2
	Qualité de la conception Analyse du CDC (DTI, ACD), algorithmes/logigrammes, choix des structures de données (structures/tableaux…).	/8
Bilan Sur 10	Qualité de la soutenance orale Contenu pertinent, respect du *timing*, réponse aux questions du Jury	/5
	Qualité du rapport final Exhaustivité, qualité de synthèse.	/5
TOTAL	Le total sur 40 points sera divisé par 2 afin d'obtenir une note sur 20.	/40

La note sera plafonnée à 20/20.

Dans le cas où un écart d'investissement dans le projet est relevé, une note individuelle remplacera la note de groupe pour l'étudiant.e concerné.e. Ainsi, quelqu'un n'ayant fourni aucun effort sur le projet pourra se voir attribuer la note de 0/20.

La plupart de ces points seront vérifiés lors de la soutenance. Si vous dépassez le temps prévu pour la soutenance, vous aurez 0 sur ce qui n'aura pas pu être vérifié lors de celle-ci.

Base demandée

Pour obtenir la moyenne à la partie code, le jeu devra permettre de faire s'affronter deux à quatre joueurs dans une arène. Chaque joueur pourra jouer lorsque c'est son tour : se déplacer là où c'est possible et attaquer les personnages se trouvant dans la zone d'action de l'attaque utilisée.

Livrables

Les livrables à fournir et délais à respecter sont précisés sur Moodle. Vous devez ainsi rendre :

- Un **rapport de conception** (**.pdf**): ce rapport doit simplement prouver que vous avez effectué une conception avant d'avoir commencé à coder. Vous y montrerez vos recherches : scans de brouillons (logigrammes…), photos de recherches au tableau, vos analyses du CDC (DTI et ACD), différents visuels... Vous expliquerez aussi comment vous allez vous répartir les tâches et la méthodologie que vous souhaitez mettre en place.
- Le **code source** de votre jeu, sans fichiers inutiles mais avec les fichiers utiles (n'en oubliez pas), est à rendre sur **GitHub et** sur Moodle. IMPORTANT : chaque membre de l'équipe doit effectuer des *commit + push* pertinents **régulièrement** afin de justifier son implication, ce n'est pas le nombre de commits qui compte mais la qualité de leur contenu. Les commits doivent être réguliers (plusieurs par semaine et par personne), si vous ajoutez tout le code source en un seul commit à la fin, nous ne considérerons pas que vous avez utilisé Git.
- **Support de votre soutenance** (**.pdf**) : vous rendrez vos slides présentées lors de votre soutenance (voir rubrique Soutenance).
- **Rapport final** (**.pdf**) : sous la forme d'une présentation (PowerPoint, Keynote…) exportée en PDF. Aucune rédaction n'est demandée mais vous devez présenter :
    - L'équipe : prénoms, noms, photos
    - Pour chaque membre de l'équipe :
        - Tâches assignées
        - Pourcentage d'accomplissement de chaque tâche
    - Votre conception (comment vous aviez imaginé réaliser le jeu),
    - Ce qui est différent de votre conception,
    - Votre organisation en équipe (outils mis en place),
    - La structure de votre application (structures, tableaux, bibliothèques, cheminements…),
    - Vos principaux algorithmes (les plus intéressants, sous la forme de logigramme par exemple),
    - La liste complète des fonctionnalités et leur pourcentage d'aboutissement : (100% : parfaitement fonctionnel, 0% : non commencé),
    - Un bilan
- Bonus : une **vidéo** où vous vous mettez en scène en présentant comment on fait un projet d'informatique : étapes de conception, d'implémentation, réunions d'équipes, étapes de débogage, de tests, soutenance, et résultat final. Cette vidéo pourra être diffusée par l'école sur les réseaux sociaux ou encore lors de journées portes ouvertes par exemple. Faites-là la plus attractive possible ! La vidéo devra comporter des sous-titres afin de pouvoir être diffusée lors de JPO.

Soutenance

Durée : 23 minutes maximum.

Vous présenterez votre travail lors d'une soutenance orale devant un jury. Elle sera découpée en 4 temps :

- Présentation (slides),
  - Organisation de l'équipe : qui a fait quoi, avec pour chaque tâche, son pourcentage d'aboutissement (100% : parfaitement fonctionnel, 0% : non commencé),
  - Choix de conception : comment vous représentez les données dans votre code source (tableaux, structures, pointeurs…),
  - Qualité de vos algorithmes : réutilisabilité, complexité, sécurisation, choix algorithmiques et techniques…
  - Qualité de votre code : syntaxe, découpe en fonctions réutilisables, en bibliothèques…,
  - Ce qui a évolué entre votre conception et ce qui a été fait dans votre code.
- Démonstration du jeu (votre démo doit être prête avant la soutenance et être rapide),
- Analyse du code par le jury (depuis l'IDE, ouvert préalablement),
- Questions personnalisées du jury.

En cas de dépassement, votre soutenance sera interrompue et vous ne serez pas évalué sur les parties non traitées. Gérez votre temps ! Préparez votre soutenance.

Plagiat

Chaque code source sera comparé à tous les autres (Paris et Lyon) via un outil spécialisé dans la détection de plagiat de code source. En cas de plagiat avéré, la note de 0/20 sera attribuée à tous les membres des deux groupes concernés, qui seront également convoqués en conseil de discipline afin de décider d'une punition et de statuer sur leur exclusion de l'école.

Bonne chance à tou.te.s !