

INTITULÉ	COEFFICIENT RELATIF CRITÈRE
	23
■ PRÉSENTATION ÉCRITE ET ORALE	8
► DIAGRAMME DE CLASSES (C'EST LE DIAGRAMME QUI EST NOTÉ ET PAS LE MODÈLE QU'IL REPRÉSENTE)	2
✓ Cohérence du diagramme : <ul style="list-style-type: none"> cohérence des relations entre elles : Leurs sens semblent cohérent — même si ce n'est pas celle du cours (Par exemple, les relations d'utilisation et d'héritage sont parfois inversées) cohérence des relations d'utilisation avec les attributs qui les représentent (pas grave s'il n'y a pas d'attributs pour représenter les relations d'utilisation), cohérence des types d'attributs (les mêmes notions sont représentées par les mêmes types) 	1
✓ Précision : <ul style="list-style-type: none"> spécification des types des attributs, indication que l'attribut représente une collection ou non spécification du sens des relations, légende explicative — si conventions différentes de celles du cours. 	2
► RESTE DU RAPPORT	4
✓ Orthographe et présentation : Il ne s'agit pas de traquer la moindre faute, ce qui serait injustement pénalisant pour certains, mais d'évaluer l'investissement dans la réalisation (passage au correcteur orthographique, cohérence de la présentation qui dénote une réflexion...)	1
✓ Niveau de détails : Le niveau de détails est mauvais quand les commentaires sont trop techniques (au pire bouts de programmes pas commentés) ou, au contraire, trop vagues. Il est bon quand il explique sobrement le rôle des méthodes avec quelques détails si besoin.	2
✓ Richesse informationnelle : Est-ce qu'il y a des explication fouillées ou tout au moins suffisante pour comprendre comment est fait le programme et quels choix ont été effectués... ?	3
✓ Jeux d'essais : Les jeux d'essais permettent de recouvrir les principales fonctionnalités (à neutraliser si la réalisation n'est pas fonctionnelle).	2
► DÉMO	2
✓ Clarté : <ul style="list-style-type: none"> Note basse : trop détaillé, trop verbeux, confus... Note haute : présentation fluide. Bon passage de parole si deux intervenants. 	1
✓ Caractère informatif : Permet bien de comprendre les fonctionnalités du programme et quelles instances/méthodes les assurent. Mode démo éventuel. Utilisation de jeux d'essais.	2
■ RÉALISATION	12
► PROGRAMMATION DE BASE	2
✓ Bases de la programmation : boucles, conditionnelles, expressions booléennes...	1
✓ Utilisation des types objets... ou pas : type String ou, pire, int, mis à toutes les sauces...	3
✓ Respect global des normes de programmation vues en cours : Utilisation des accesseurs ou au moins de this pour les attributs, choix des noms d'attributs, de méthodes, de classes...	1
► DÉCOUPAGE	7
✓ Choix des classes : Éviter les classes anecdotiques, qui n'ont pas de rôle fonctionnel ou de réalité conceptuelle (Les notions qu'elles représentent sont explicitement nommées dans l'analyse du problème), le mieux étant qu'elles aient les deux.	1

<p>✓ Choix des attributs et des méthodes et modalités d'accès : Les attributs doivent représenter des propriétés des instances et pas seulement des variables globales qui contiennent de l'information temporaire ou pas, les méthodes doivent traduire l'accès à l'objet ou son comportement. Bonus si les méthodes d'accès à des attributs qui ne doivent pas être modifiés et celles qui correspondent à des fonctionnalités annexes (initialisation d'un objet par exemple) ne sont pas publiques.</p>	1
<p>✓ Positionnement des attributs et méthodes : La question se pose quand une fonctionnalité met en jeu des instances de plusieurs classes. L'emplacement des attributs et méthodes qui la représentent dans telle ou telle classe peut alors être plus ou moins judicieusement choisi.</p>	2
<p>✓ Délégation : La délégation est incorrecte quand certaines classes ne contiennent que de l'information sans avoir de rôle propre, quand la prise en charge des fonctionnalités est centralisée, globalisée, monolithique alors qu'elle aurait pu être découpée en fonctionnalités plus simples prises en charge par des instances/méthodes spécialisées. Par exemple, le toString — ou l'affichage — du plateau doit faire appel à celui des cases. Dans l'idéal, presque toutes les instances jouent un rôle actif ; elles ne doivent pas se limiter à contenir de l'information.</p>	2
<p>✓ Héritage : <ul style="list-style-type: none"> • Note basse : utilisation a minima ou non pertinente, • Note haute : utilisation parfaitement adaptée au problème. </p>	1
► RÉALISATION DES FONCTIONNALITÉS PRINCIPALES	3
✓ Programme complètement/partiellement/pas du tout fonctionnel ?	1
<p>✓ Efficacité algorithmique : Il s'agit surtout, par exemple, de valoriser les stratégies de recherche d'information qui tendent à se rapprocher d'un accès direct en utilisant un type objet, par rapport à d'autres stratégies qui nécessitent des parcours de systèmes de structures (tableau ou ArrayList)</p>	1
<p>✓ Résolution des principaux problèmes : Qualité de la résolution des difficultés principales : génération de la grille des cases, gestion du jeu, déplacement du joueur et des pierres, fin de jeu...</p>	1
■ BONUS	3
► FONCTIONNALITÉS SUPPLÉMENTAIRES (PLUSIEURS JOUEURS, AUTRES TYPES DE CASES...)	2
<p>✓ Quantité des apports : Plus ou moins d'innovations par rapport à la problématique de l'énoncé</p>	1
<p>✓ Qualité des apports : Innovations complexes ou anecdotiques</p>	3
► ÉTUDE DES SITUATIONS DE BLOCAGE (IMPOSSIBILITÉ D'ATTEINDRE L'ARRIVÉE)	3
<p>✓ Approfondissement de la réflexion : Investissement dans cette étude, nombre de situations étudiées</p>	3
<p>✓ Pertinence de la réflexion : Qualité de la typologie des situations distinguées et des idées proposées éventuellement comme solution pour les différents cas</p>	2