SUBOO : Sorbonne Université Build Order Optimizer

Dans un jeu de stratégie temps réel (STR) comme StarCraft II, le joueur doit trouver un équilibre économique entre la construction d'ouvriers (qui augmentent le revenu) la construction de bâtiments de technologie (qui débloquent de nouvelles unités) et la construction d'unités d'attaque (pour gagner la partie).

Le joueur démarre avec un Centre, une banque constituée de 50 PO (Pièces d'Or) et 5 ouvriers.

Le jeu considère deux types de ressources : l'or et la nourriture. A chaque instant le joueur dispose d'une banque constituée des PO qu'il a récolté et économisé, et un total de nourriture N qui correspond à la quantité maximum d'unités que le joueur peut construire avec les bâtiments qu'il possède.

Chaque unité du jeu possède des caractéristiques qui lui sont propres (cf tableau). On considère une version simplifiée du jeu avec un nombre limité de types d'unités.

Les ouvriers construisent des fermes pour augmenter la nourriture disponible, des casernes qui permettent de produire des soldats, et (en général un seul) Hall sans lequel la caserne ne peut pas construire des Boss. Quand ils ne sont pas mobilisés par une construction, ils produisent 1PO par ouvrier et par unité de temps. Les soldats et les Boss sont des unités d'attaque, le but du jeu est d'en construire beaucoup et vite.

Chaque unité ou bâtiment a un coût de construction (or et nourriture). Pour engager la construction d'une unité il faut posséder le coût indiqué dans la table en PO et en nourriture N.
Ce coût en ressources est consommé dès que la production de l'unité est lancée. Mais l'unité démarre *en construction* pendant son « Tps Cstr ». Les unités coûtent donc de la nourriture même si elles sont encore en construction. Les bâtiments (Centre, Ferme, Caserne, Hall) ne coûtent pas de nourriture, mais les Fermes et Centre en construction ne contribuent pas de nourriture N.

Catégorie	Unité	Cout PO	Cout N	Tps Cstr	Prérequis	Mobilise	Fournit
Batiment	Centre	300		200	Ouvrier	Ouvrierx200	+6N
	Ferme	100		60	Ouvrier	Ouvrierx60	+6N
	Caserne	150		100	Ouvrier,Ferme	Ouvrierx100	
	Hall	200		120	Ouvrier,Caserne	Ouvrierx120	
Unité	Ouvrier	50	1	30	Centre	Centrex30	1PO/s
	Soldat	100	1	50	Caserne	Casernex50	
	Boss	200	3	100	Hall,Caserne	Casernex100	

Certaines unités ont des prérequis, par exemple il faut une ferme pour produire une caserne, et il faut un Hall pour que la caserne puisse produire des Boss. Les prérequis sont satisfaits dès qu'une unité du type demandé est présente. De plus, chaque construction d'unité mobilise un ouvrier ou un bâtiment pendant une certaine durée.

On considère que les unités dans une partie peuvent être dans l'un des trois états suivants :

- *en construction* les unités nouvellement crées sont dans cet état pendant une durée déterminée par leur temps de construction. L'unité devient *disponible* à la fin de son temps de construction.

- *mobilisé* les bâtiments qui construisent une unité sont mobilisés, de même que l'ouvrier qui construit le bâtiment. L'unité redevient *disponible* quand elle finit sa mobilisation.
- *disponible* si une unité n'est ni en construction, ni mobilisée, c'est qu'elle est disponible. On considère que dans cet état les ouvriers génèrent naturellement 1PO par unité de temps. Seules les unités *disponibles* peuvent être mobilisées pour une construction.

Pour se développer on peut construire de nouveaux ouvriers dans le Centre mais on sera vite limité par la nourriture N. Il faut donc construire des Fermes pour entretenir une population plus importante. Si on atteint le seuil de nourriture, on ne peut plus construire de nouvelles unités jusqu'à ce que la situation soit améliorée.

Un *état* d'une partie est caractérisé par la banque (PO disponible) et par les unités que le joueur possède à cet instant. Pour chacune des unités, le jeu connait son état à cet instant et le temps restant avec qu'elle ne devienne disponible si elle ne l'est pas.

Le jeu est mis à jour à chaque unité de temps ; à cette étape les ouvriers disponibles (qui récoltent donc) ajoutent 1 PO par ouvrier à la banque. On comptabilise les ressources en nourriture disponible à partir du décompte des bâtiments terminés (mobilisé ou non) présents.

On peut prendre à cet instant une décision qui consiste à produire une unité si les prérequis spécifiques à l'unité sont satisfaits, si la banque le permet, et que la limite de nourriture le permet. Le jeu boucle en interrogeant la prise de décision à chaque pas de temps.

Un ordre de construction (ou Build Order BO) est une séquence d'actions de construction. Un BO est réalisable si on peut l'exécuter en respectant les contraintes du jeu décrites au-dessus en attendant une durée arbitraire entre chaque action (i.e. les prérequis sont satisfaits). Son temps d'exécution (idéal) est calculé avec l'attente la plus courte possible entre chaque action.

Notre application devra construire le BO réalisable le plus rapide possible pour atteindre un objectif donné. L'objectif est défini par un certain nombre d'unités de chaque type (qu'on souhaite obtenir) et/ou une banque à atteindre.

Par exemple, obtenir 5 Boss et 12 attaquants le plus vite possible. Ou obtenir 2000PO en banque le plus vite possible.

On demande que SUBOO puisse supporter plusieurs versions du jeu : l'arbre de technologie et le prix des unités varient régulièrement au fil des versions du jeu. L'utilisateur doit pouvoir configurer un objectif, défini par un certain nombre d'unités de chaque type et une banque cible. Avec cet objectif, il peut obtenir un BO naïf (mais réalisable) qui permet de l'atteindre : seuls les éléments strictement nécessaires sont produits. Réciproquement, en fournissant un BO, l'utilisateur doit savoir s'il est réalisable et calculer sa durée idéale. S'il n'est pas réalisable, l'outil doit proposer une correction du BO (en ajoutant des actions). Enfin l'utilisateur doit pouvoir obtenir un BO optimisé, qui minimise le temps pour atteindre un objectif. L'utilisateur veut le meilleur BO possible, même si le calcul est coûteux. Enfin, l'outil devra pouvoir interagir avec une IA qui joue au jeu en temps réel. Dans ce cadre l'IA fournit régulièrement pendant la partie une observation de l'état actuel du jeu (au lieu de l'état initial) et un objectif qu'il faut optimiser rapidement (prise de décision court terme). Ces échanges doivent utiliser une API et non une IHM.