PLAN DE TEST
Intelligence Artificielle

Robot Lejos EV3

Adam Cotard

Quentin Juillat

Tom Laucournet

Brice Mc Carthy

Tao Thill

Classe	Méthode	Description	Réussite	Changements éventuels
• **package moteurs**				
Pince	void ouvrir()	ouverture complète des pinces		Nous avons rajouté une vérification de l'état actuel des pinces avant de les ouvrir ou de les fermer avec un "if (estOuvert())" pour empêcher l'ouverture si elles sont déjà ouvertes et la fermeture si elles sont déjà fermées.
Pince	void fermer()	fermeture complète des pinces		
Pince	boolean estOuvert()	return true lorsque les pinces sont ouvertes		On a commencé par faire la méthode estFermée() à l'aide d'un angle mais nous avons changé de méthode
Pince	void fermerObligatoirement()	Ferme les pinces de manière forcée, indépendamment de l'état actuel des pinces.		Cette méthode a été ajoutée en cas de problème avec l'état interne des pinces.
Pince	void setOuvert(boolean b)	Définit l'état des pinces comme étant ouvert ou fermé.		RAS
Pince	boolean estOuvert()	Vérifie si les pinces sont ouvertes		RAS
Action	void avancer()	Fait avancer le robot		RAS
Action	void avancer (int speed)	Fait avancer le robot à la vitesse donnée		RAS
Action	void avancer (int distance, int speed)	Fait avancer le robot à la vitesse donnée et de la distance donnée		RAS

Action	void avancerAsync (int distance)	Fait avancer le robot de manière asynchrone d'une distance mise en paramètre, en étant asynchrone il continue l'exécution du programme sans attendre de ne plus avancer		Nous avons créer cette méthode en parallèle de avancer() afin d'optimiser certaines actions qui pouvaient se faire tout en continuant à avancer (exemple : ouvrir les pinces sans avoir à s'arrêter)
Action	void setVitesse(int vitesse)	Défini la vitesse de déplacement linéaire du robot		RAS
Action	void reculer()	Fait reculer le robot		RAS
Action	void reculer (int distance, int vitesse)	Fait reculer le robot à la vitesse donnée et de la distance donnée		RAS
Action	void stop()	arrête le robot		RAS
Action	void tournerD (int angle)	Fait tourner le robot vers la droite de l'angle en paramètre		Un coefficient présent dans la méthode doit souvent changer afin de corriger les erreurs incontrôlable du robot (le moteur d'une roue moins puissant que l'autre, l'écartement des roues qui peut bouger pendant les expérimentations, etc…)
Action	void tournerG (int angle)	Fait tourner le robot vers la gauche de l'angle en paramètre		Même changement potentiel que pour la méthode tournerD(int angle)
Action	void asyncTournerG (int angle)	Fait tourner le robot vers la gauche de l'angle en paramètre de manière asynchrone		Cette méthode a été créée essentiellement pour la méthode recherche afin de pouvoir capturer des distances et les comparer parallèlement à la rotation du robot. Mais également le même changement potentiel que pour la méthode tournerD(int angle)

Action	void asyncTournerD (int angle)	Fait tourner le robot vers la droite de l'angle en paramètre de manière asynchrone		Même changement potentiel que pour la méthode tournerD(int angle)
Action	boolean ismoving()	Vérifie si le robot est en mouvement		RAS
Action	void seDecalerD()	Le robot tourne de 45 degrés vers la droite, avance de 30cm puis se remet droit. Ce décalage a pour but d'éviter les palets étant sur la même ligne que le palet qui est en train d'être ramené		RAS
Action	void seDecalerG()	Le robot tourne de 45 degrés vers la gauche, avance de 30cm puis se remet droit. Ce décalage a pour but d'éviter les palets étant sur la même ligne que le palet qui est en train d'être ramené		RAS
• **package capteurs**				
CouleurCapteur	Color getCouleur()	Récupère la couleur détectée par le capteur sous forme d'un objet Color. Cette méthode effectue une lecture des échantillons RGB et les convertit en un Color.		RAS
CouleurCapteur	String nomCouleur (int	Retourne le nom de la couleur en		RAS

	r, int g, int b)	fonction des valeurs RGB données		
CouleurCapteur	void fermerCapteurCouleur ()	Ferme le capteur couleur afin d'économiser de la batterie		RAS
DistanceCapteur	void ouvrir()	Ouvre le capteur de distance		RAS
DistanceCapteur	void fermer()	Ferme le capteur de distance		RAS
DistanceCapteur	float getDistance()	Retourne la première valeur du tableau des échantillons de distance		RAS
DistanceCapteur	float[] recherche(float[] tab)	Cette méthode ajoute la distance courante capturée dans le tableau passé en paramètre		RAS
DistanceCapteur	void setSample ()	Remplit le tableau sample avec les données actuelles du capteur à ultrasons.		RAS
DistanceCapteur	float[] getSample ()	Retourne le tableau des échantillons de distance		RAS
DistanceCapteur	SampleProvider getDistanceSampler()	Retourne le fournisseur d'échantillons permettant d'obtenir les mesures de distance.		RAS
DistanceCapteur	EV3UltrasonicSensor getCapteurDistance()	Retourne l'instance du capteur à ultrasons.		RAS

ToucherCapteur	boolean estTouche()	Vérifie si le capteur de contact est actuellement touché, retourne “true” si le capteur est pressé, “false” sinon		RAS
• **package iapackage**				
Principal	void recalibrage()	Recalibre le robot afin qu’il soit face au but adverse		RAS
Principal	boolean avancerPlusPrendrePalet(int vitesse, int distance)	Le robot avance à la vitesse donnée tout en ouvrant ses pinces jusqu'à ce que le capteur de contact soit pressé par un palet ou que la distance en paramètre soit atteinte, puis ferme ses pinces. Cette méthode return “true” si un palet a été attrapé, “false” sinon.		Cette méthode a été créée pour la méthode recherche() afin de corriger les mauvaises orientations du robot après avoir capté un palet. Ainsi si cette méthode renvoie “false” après qu’elle soit lancée en direction du palet capté, c’est que celui-ci a pas été trouvé et on peut donc relancer une recherche.
Principal	void avancerPlusPrendrePaletAsync(int vitesse)	Le robot avance à une vitesse donnée jusqu'à ce que le capteur de contact soit pressé par un palet ou qu’une ligne blanche soit atteinte, puis ferme ses pinces		On pourrait rajouter une utilisation du capteur de distance pour vérifier que le palet est toujours là pendant qu’on s’y dirige et ainsi éviter de devoir aller jusqu’à la ligne blanche.

Principal	void avancerPlusPrendrePaletAsyncEtOuvre(int vitesse)	Le robot avance à une vitesse donnée tout en ouvrant ses pinces jusqu'à ce que le capteur de contact soit pressé par un palet ou qu'une ligne blanche soit atteinte, puis ferme ses pinces		On pourrait rajouter une utilisation du capteur de distance pour vérifier que le palet est toujours là pendant qu'on s'y dirige et ainsi éviter de devoir aller jusqu'à la ligne blanche.
Principal	void avancerJusqueCouleur(String c)	Le robot avance jusqu'à ce qu'il détecte la couleur passée en paramètre sous son capteur couleur.		RAS
Principal	void avancerJusqueLigne()	Le robot avance jusqu'à ce qu'il détecte une ligne de couleur noire, jaune ou rouge. Cette méthode est utile pour se décaler d'une case sur la table		RAS
Principal	void eviterRobot()	Permet à notre robot d'éviter un robot adverse si il est en face de lui		Le capteur distance a parfois des difficultés à détecter la présence d'un robot, cette méthode n'est donc pas optimal et ne fonctionne pas souvent.
Principal	void recherche ()	Le robot recherche un palet proche en faisant un tour sur lui-même et en s'arrêtant lors de la détection d'un palet. Une fois détecté, le robot va se diriger vers celui-ci en lançant la méthode avancerPlusPrendrePalet(int vitesse, int distance) (la distance en paramètre est égale à la distance		Cette méthode a beaucoup changé tout au long du projet. Initialement le robot faisait un tour complet puis comparait toutes les distances obtenues pour garder la plus petite et se retourner vers elle après avoir vérifié que les dimensions correspondaient à celles d'un palet. Finalement nous sommes partis sur cette idée, qui consiste à s'arrêter dès la première détection d'un palet pour ne pas perdre du temps à faire

		du palet capté + 30cm). Si aucun palet n'est touché, la méthode se relance.		un tour complet. Cependant la détection des palets demeure instable malgré beaucoup d'optimisations et temps consacré à cette méthode.
Principal	void case345(boolean droite)	Méthode utilisée au lancement de la partie dans le cas des stratégies 3, 4 et 5 (stratégie de vol d'un palet sur la ligne adverse). Cette méthode prend le premier pale face à nous, le ramène dans le but adverse, puis le robot se dirige vers la case où le robot adverse a commencé pour plus tard pouvoir voler un palet de sa ligne de départ. Le booléen en paramètre détermine la direction vers laquelle notre robot va se décaler en fonction du placement de l'équipe adverse.		RAS
Principal	void test3palets1(boolean droite)	Méthode utilisée au lancement de la partie dans le cas des stratégies autres que 3, 4 et 5. Cette méthode fait en sorte de ramener les 3 palets face à nous au départ dans le camp adverse. Le booléen en paramètre détermine la direction vers laquelle notre robot va se décaler en fonction du		RAS

		placement de l'équipe adverse.		
Principal	static double tempsVersAngle(long t)	Calcule l'angle associé à un temps de rotation donné. Cette méthode est utilisée dans la méthode recherche() quand le robot tourne sur lui-même jusqu'à repérer un palet. Elle instancie ainsi à chaque fois l'angle du robot pour qu'il puisse par la suite se recalibrer vers le camp adverse avec la méthode recalibrage()		Quand le robot a effectué plusieurs recherches sans trouver de palet, la précision du recalibrage peut être biaisée.