## function seconde\_to\_temps(t: LongInt): String;

Transforme le temps (stocké comme un entier, en centième de seconde) en une chaîne de caractère au format (m minutes, s secondes, ccc centièmes).

On obtient la valeur des minutes, secondes et centièmes via les opérateurs modulo et diviseur, et on assure le format de sortie via l’ajout éventuel de 0 pour les minutes et secondes.

## procedure ajouter\_physique(var physique: T\_PHYSIQUE\_TABLEAU);

Ajoute un nouvel élément physique au tableau.

La procédure sauvegarde l’ancien pointeur, demande au système l’allocation d’un bloc de mémoire d’une taille supérieure de 1 par rapport au précédent, recopie via une boucle l’adresse des anciens éléments physiques dans le nouveau bloc, demande et stocke l’adresse du nouvel élément physique dans la dernière case (la ‘nouvelle’ case) et initialise ces valeurs à 0. La procédure libère alors l’ancien bloc de pointeurs, et augmente la taille du tableau de 1.

## procedure ajouter\_enfant(var element: T\_UI\_ELEMENT);

Ajoute un enfant à un élément d’UI.

La procédure sauvegarde l’ancien pointeur, demande au système l’allocation d’un bloc de mémoire d’une taille supérieure de 1 par rapport au précédent, recopie via une boucle l’adresse des anciens éléments physiques dans le nouveau bloc, demande et stocke l’adresse du nouvel enfant dans la dernière case (la ‘nouvelle’ case) et initialise ses valeurs. La procédure libère alors l’ancien bloc de pointeurs, et augmente le nombre d’enfants de 1.

## function isInElement(element: T\_UI\_ELEMENT; x, y: Integer): Boolean;

Teste si un élément d’UI se situe au coordonnées indiquées (référentiel fenêtre)

En premier, la procédure soustrait récursivement l’ensemble des coordonnées x,y des parents de l’élément au coordonnées des positions demandées, afin de pouvoir par la suite tester si celles-ci se situent dans le rectangle x,x+w,y,y+h de l’élément cible.

## function hitBox(surface: PSDL\_Surface; p: SDL\_Rect; a: Real; colors: array of TSDL\_Color): T\_HITBOX\_COLOR;

Compare les couleurs d’un rectangle de 12 points (cf types) , d’état p et d’angle a, d’une surface avec les couleurs fournies, renvoie un T \_HITBOX\_COLOR, liant chaque couleur trouvée avec son point.

La fonction initialise d’abord les données nécessaires, en calculant par exemple les fonctions trigonométriques afin d’optimiser les performances. Elle va ensuite tester chacun des 12 points : le point numéro 1 d’abords, puis 2 bande de 5 points de chaque coté ( coté de la voiture) et enfin le point arrière. La couleur de chaque point est récupérée via pixel\_get, testée via hitBoxInList, et enfin ajouté au tableau de sortie via hitBoxAddList.

Cette fonction est celle des collisions, en effet elle permet de tester sur quel ‘sol’ est notre voiture : on teste la couleur des points sur l’image du circuit.

## function pixel\_get(surface: PSDL\_Surface; x,y: Integer): TSDL\_Color;

Renvoie la couleur d’un pixel sur une surface

La fonction commence par bloquer la surface si cela est requis, récupère le pointeur vers les pixels de la surface SDL : Dans SDL, les pixels d’une surface de w\*h sont stockés sous forme d’un tableau a 1 dimensions, de taille w\*h. On ensuite lire le pixel correspondant à la coordonnée demandée. Le pixel est stocké comme un entier sur 4 octets, 3 pour chaque couleurs, et 1 pour la transparence (alpha). On convertit cette entier en TSDL\_COLOR (r,g,b,a).

Nous avons rencontré des problèmes de boutisme avec cette fonction, en effet, les architectures x86(-64) ou ARM utilisé sur nos ordinateurs stockent les bits de poids faible en tête, ce qui a pour effet d’inverser la valeur de r et b. On utilise la compilation conditionnelle permise par pascal afin d’inverser ces 2 valeurs. On débloque la surface a la fin de la lecture.

## function hitBoxInList(c: TSDL\_Color; colors: array of TSDL\_Color): Boolean;

Teste si une couleur est présente dans une liste de couleurs.

Cette fonction boucle simplement dans une liste de couleurs et renvoie vrai si elle a trouvé une couleur spécifique passée en paramètre.

## procedure hitBoxAddList(var l: T\_HITBOX\_COLOR; n: ShortInt; c: TSDL\_COLOR);

Ajoute un élément a un T\_HITBOX\_COLOR.

La procédure augmente la taille du tableau de 1 et initialise la case avec le numéro du point et la couleur passé en paramètre.

## function isSameColor(a: TSDL\_Color; b: TSDL\_Color): Boolean;

Compare 2 couleurs

Ici, on renvoie vrai uniquement si les 3 valeurs de couleurs sont identiques ( gain de lisibilité dans le code)

## function ZoomMin(a,b: Real): Double;

Renvoie le minimum borné entre deux valeurs.

Renvoie min(min(a,b),0.3,1)

Voir afficher\_camera

## procedure imageLoad(chemin: String; var surface: PSDL\_Surface; alpha: Boolean);

Charge et transforme une image png en surface prête a l’affichage.

La procédure charge l’image via la librairie sdl\_image, et transforme celle-ci via displayFormat ( avec un masque alpha ou non). Cette transformation permet a l’image d’être adapté au format d’affichage de l’écran, on augmenter ainsi les performances.

## procedure updatePseudo(k : TSDLKey; var pseudo: String);

Met a jour un pseudo ( chaîne de caractère) en fonction d’un appui clavier.

La procédure récupère d’abords l’état actuel du clavier, c’est a dire un tableau de booléen correspondant a chaque touche. Elle teste ensuite suivant les valeurs de la table ASCII la touche (k) pressée et ajoute cette lettre au pseudo (a-z, A-Z avec shift, 0-9).

Les claviers sous Windows renvoyant toujours des valeurs correspondant a un clavier qwerty, on utilise la compilation conditionnelle permise par Pascal afin d’ajouter la bonne lettre.

## procedure scoreMaj(fichier: String; miseAJour: T\_SCORES);

Met a jour le fichier de score en fonction des temps réalisés.

La fonction va d’abords lire le contenue du fichier (scoreLire), puis initialiser un tableau de booléen (dont la taille est le nombre joueurs/temps).

Par la suite, on boucle dans chaque score du fichier, on test si l’utilisateur est le même (on bascule l’indice de booléen a vrai) et si son temps est inférieur, si cela est le cas, on modifie le tableau de scores.

Si a la fin de la boucle dans le fichier on a pas trouvé tous les utilisateurs ( c’est-à-dire que l’on a un nouvelle utilisateur) on ajoute leurs temps a la fin du tableau de score.

Enfin, on réécrit le fichier de score (scoreEcrire)

## procedure scoreLire(nomFichier: String; var scores: T\_SCORES);

Lit le contenu d’un fichier score et renvoie un tableau T\_SCORES

La fonction assigne a une variable le fichier, positionne le curseur de lecture au début, et lit chaque score afin de remplir un tableau T\_SCORES de même taille, avant de fermer le fichier.

## procedure scoreEcrire(nomFichier: String; scores: T\_SCORES);

Écrit un variable T\_SCORES dans un fichier

La fonction assigne a une variable le fichier, l’ouvre en écriture et écrit chaque score dans le fichier avant de le refermer.

## function min(liste : array of LongInt): LongInt;

Renvoie le minimum d’un tableau d’entier

## procedure freeUiElement(var element: P\_UI\_ELEMENT);

Libère la mémoire récursivement d’un T\_UI\_ELEMENT

La procédure assigne la valeur 0 au tableau d’enfants, puis libère la surface de l’élément via SDL\_FreeSurface, libère la police via TTF\_Close, et enfin détruit l’élément en libérant l’élément.

## procedure freeInfoPartie(var infoPartie: T\_GAMEPLAY);

Libère la mémoire de T\_GAMEPLAY

On libère la surface de la map (SDL\_FreeSurface), le skin de chaque joueurs, et enfin on détruit le tableau de joueurs.

## procedure getBestScore(var scores: T\_SCORES);

Algorithme de tri sur T\_SCORES ( tri par insertion)

## procedure frame\_afficher\_low(var element: T\_UI\_ELEMENT; var frame: PSDL\_Surface; etat: T\_RENDER\_ETAT);

Cette procédure crée la surface à afficher sur l’écran a partir d’un T\_UI\_ELEMENT (fenetre) récursivement.

Cette fonction est une mise en application de la procédure SDL\_BlitSurface : elle empile les surfaces afin d’en créer une seul (frame), qui sera affichée sur l’écran. On va d’abord appliquer les paramètres à nos surfaces ( couleur pour un type ‘couleur’, texte pour un type ‘texte’ ), et ensuite on applique les styles aux surfaces (transparence), puis on colle celle-ci sur la surface principale (frame)

La position sur laquelle on colle la surface est passée récursivement et incrémentée à chaque fois.

Si un élément est de type texte, on incrémente de plus cette position avec la largeur du texte, afin de pouvoir positionner des éléments au bout de celui-ci (utile pour les curseurs des textes par exemple)

## procedure frame\_afficher(var element: T\_UI\_ELEMENT);

Cette procédure initialise et prépare frame\_afficher\_low

On initialise les valeurs de la position de collage à 0 puis on lance frame\_afficher\_low.

## procedure afficher\_hud(var infoPartie: T\_GAMEPLAY);

Affiche le HUD

On modifie la valeur du temps global, du numéro de tour actuel, du premier joueur, de la vitesse et du temps pour le secteur de chaque joueur.

## procedure afficher\_camera(var infoPartie: T\_GAMEPLAY; var fenetre: T\_UI\_ELEMENT);

Gère l’affichage de la caméra (mouvements et échelle)

La fonction calcule d’abord l’échelle (zoom) adapté a la situation ( 1 / 2 joueurs) :

Après avoir déterminer le centre de l’écran ( référentiel map ), on impose une constante de distance (%) entre la voiture et le bord de l’écran en hauteur et largeur. On calcul le zoom nécessaire a avoir cette distance, et on va créer une map de la bonne taille a l’aide de rotozoomSurface a partir de la map de base ( infopartie.map.base ). On prend soin de libérer l’ancienne map et de sauvegarder le zoom ( on ne crée pas une map a chaque frame, seulement si le zoom change suffisamment (via arrondi). En réalité, ce n’est pas la camera qui bouge, mais l’ensemble du circuit.

On va ensuite placer les joueurs, pour cela, on libère leur ancienne surface, on en crée des nouvelles (rotozoom) et on les placent a leur position (changement de référentiel).

## procedure course\_afficher(var infoPartie: T\_GAMEPLAY; var fenetre: T\_UI\_ELEMENT);

Cette procédure génère une image a afficher.

On appelle successivement afficher\_camera puis afficher\_hud

## procedure course\_gameplay(var infoPartie: T\_GAMEPLAY);

Gère les évènements de jeu

On définit les couleurs de map importantes ( lignes de départ / Secteurs), puis on va tester si un joueur passe ces lignes via la procédure hitbox. En fonction de la ligne, on incrémente son nombre de tour, son secteur, et on détecte la fin de partie.

## procedure frame\_physique(var physique: T\_PHYSIQUE\_TABLEAU; var infoPartie: T\_GAMEPLAY);

Calcule la physique du jeu a chaque boucle

On calcule la physique du jeu a chaque frame.On teste pour chaque joueur les collisions avec l’herbe du circuit, et on applique suivant le résultat un coefficient de frottement différent ( air ou terre ) a sa vitesse.

Ensuite, on va calculer la position de tout les éléments physique, en ajoutant dt\*vitesse a notre position

## procedure course\_user(var infoPartie: T\_GAMEPLAY);

Convertie les entrées utilisateur en valeurs de infoPartie.

On lit les événements SDL, (quitter pour la croix), et on incrémente les vitesse des joueurs / leur angle en fonction de la touche pressée. La touche P permet de forcer l’arrêt de la partie.

## procedure course\_arrivee(var infoPartie: T\_GAMEPLAY; var fenetre: T\_UI\_ELEMENT);

Affiche le tableau de fin de partie.

On cache le HUD, on remplit un tableau T\_SCORES, on récupère les meilleurs scores du circuits et enfin on affiche la surface.   
On boucle ensuite en attendant une validation par l’utilisateur, auquel cas on met a jour le fichier de score.

## procedure course\_depart(var infoPartie: T\_GAMEPLAY; var fenetre: T\_UI\_ELEMENT);

On génère une image du circuit/voitures seulement (afficher\_camera), on cache le HUD, on effectue un décompte en affichant les feux tricolore, puis on libère les surfaces et enfin on ré-affiche le HUD.

On actualise ensuite les temps de début de partie et last (dernier temps) a la valeur actuelle de la SDL

## procedure partie\_course(var infoPartie: T\_GAMEPLAY; var physique: T\_PHYSIQUE\_TABLEAU; var fenetre: T\_UI\_ELEMENT);

Cette procédure contient la boucle principale de la course :

en premier, on lance la procédure de départ (couse\_départ), puis on boucle sur les procédures suivantes :

1)On met a jour le temps du jeu (dt l’interval depuis la dernière boucle), last le temps actuel

2)On récupère les entrées utilisateurs : course\_user

3)On met a jour la physique : frame\_physique

4)On met a jour les informations de la partie : course\_gameplay

5)On génère une image a afficher : course\_afficher

6)On affiche cette image : frame\_afficher

Et enfin on calcule le temps de a attendre afin de maintenir une cadence la rafraîchissement de l’écran C\_REFRESHRATE

## procedure partie\_init(var infoPartie: T\_GAMEPLAY; var physique: T\_PHYSIQUE\_TABLEAU; var fenetre: T\_UI\_ELEMENT);

On initialise les données de la partie.

On initialise infoPartie, la couleur de la fenêtre, on charge la map dans infoPartie, les skins des joueurs ainsi que leurs données :temps secteurs, nbTour, position (en fonction du circuit) et le HUD.

## procedure jeu\_partie(config: T\_CONFIG; fenetre: T\_UI\_ELEMENT);

Procédure correspondant a une partie.

On lie la config à infoPartie, on initialise via partie\_init et on lance une partie avec partie\_course

En fin de partie, on libère les surface et infoPartie avec freeUiElement et freeInfoPartie

## procedure jeu\_menu(fenetre: T\_UI\_ELEMENT);

Affiche le menu de configuration de la partie

On charge les texture des menu roulant au préalable afin d’éviter toute fuite mémoire, on initialise les couleurs, les positions et les enfants dans fenêtre. Enfin, on affiche celles-ci.

Lorsqu’un utilisateur clic sur un élément, on met a jour sa valeur, et on teste chaque élément a la fin de la boucle afin de mettre a jour l’affichage / la configuration en fonction des choix de l’utilisateur.

Lorsque l’utilisateur valide sa partie ( entrée ) on initialise la config et on lance jeu\_partie.

## procedure tutoriel(fenetre: T\_UI\_ELEMENT);

Affiche le tutoriel

On lit dans un fichier le tutoriel, et on affiche les images correspondantes au numéros de ligne dans le texte.

## procedure score(fenetre: T\_UI\_ELEMENT);

Affiche les records de score par circuit.

Lit les données a l’aide de scoreLire, et getBestScore, puis les affiche dans des panneaux ( 1 par circuit)

## procedure menu(var fenetre: T\_UI\_ELEMENT);

Menu principal

Affiche 4 boutons correspondants au fonctions principales du jeu.

## function lancement(): T\_UI\_ELEMENT;

Initialise SDL

Initialise et test le lancement de la librairie SDL (lance la fenetre)

## A0

Initialise la fenetre ( lancement) puis lance le menu

en fin de jeu, libère toutes les surfaces, libère la librairie TTF et la SDL.