ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΕΧΝΟΛΟΓΙΑ ΑΝΑΠΤΥΞΗΣ ΕΥΦΥΩΝ, ΠΟΛΥΜΕΣΙΚΩΝ ΚΑΙ ΚΙΝΗΤΩΝ ΔΙΕΠΑΦΩΝ – ΗΥ454

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2006
ΔΙΔΑΣΚΩΝ: ΑΝΤΩΝΙΟΣ ΣΑΒΒΙΔΗΣ
ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ - PROJECT

Γενικά

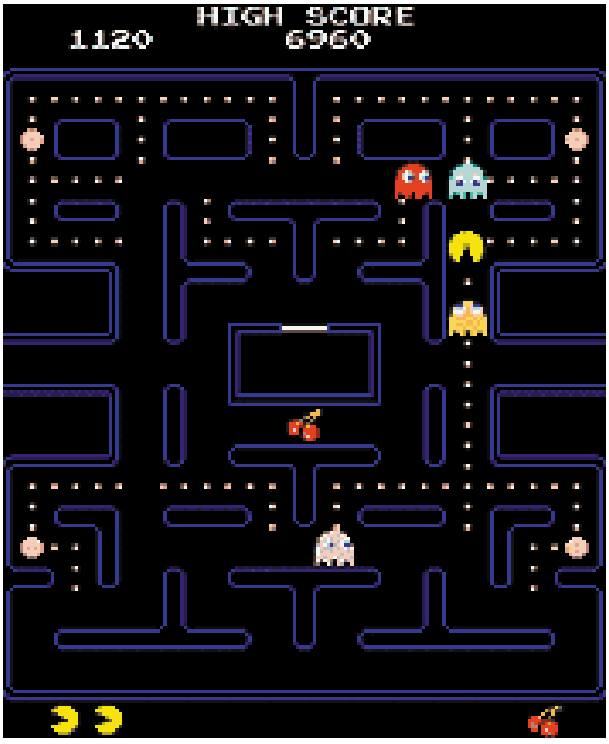
Η κατασκευή του κλασικού παιχνιδιού γνωστό με το όνομα "Pac Man" που πρωτοπαρουσιάστηκε το 1980, σχεδιασμένο από τον Moru Iwatan της Namco. Τα χαρακτηριστικά του παιχνιδιού αυτού είναι:

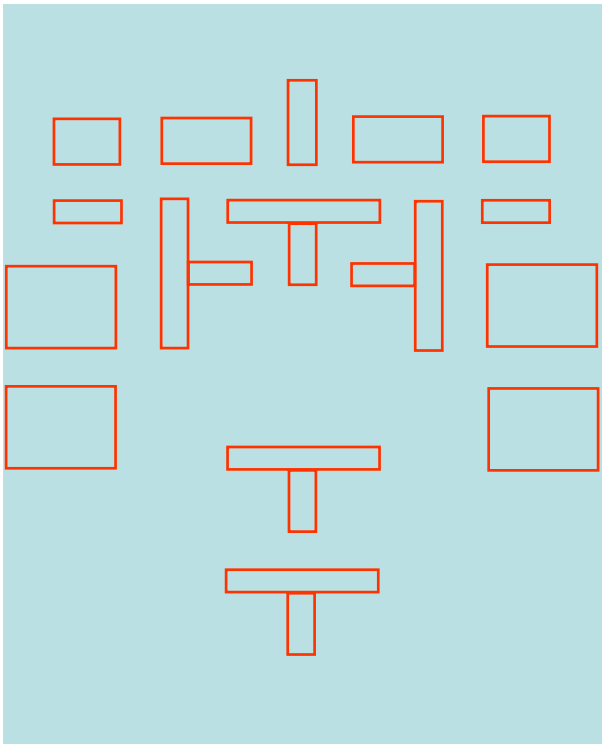
- Ένα ή δύο βασικά επίπεδα (για το Project) με ένα terrain το οποίο έχει φυσικό μέγεθος όσο η οθόνη.
- Δεν είναι tile-based game, αλλά έχει ένα καλά δομημένο terrain που οριοθετεί κυρίως την κίνηση του βασικού χαρακτήρα (Pac Man), των αντιπάλων (Ghosts) και των αντικειμένων επιβράβευσης (Bonus objects, Fruits).
- Αντίπαλοι οι οποίοι κινούνται «έξυπνα» ώστε να πλησιάσουν το βασικό χαρακτήρα, ή να τον αποφύγουν όταν έχει τη δυνατότητα να τους «καταναλώσει».
- Ειδικά bonus items που δίνουν προσωρινά τη δυνατότητα στον Pac Man να καταναλώνει («τρώει») τους αντιπάλους του.
- Ο κυρίαρχος χαρακτήρας κινείται σε τέσσερις κατευθύνσεις με σταθερή ταχύτητα, ενώ δεν έχει κανένα είδος ειδικού όπλου, εκτός από την περιστασιακή ικανότητα να κατατροπώνει τα Ghosts για περιορισμένο χρονικό διάστημα (upon collision).

Θα ψάξετε στο δίκτυο και θα βρείτε πολλές περιπτώσεις υλοποίησης του παιχνιδιού τα οποία θα παίξετε για να δείτε ώστε να αναπαράγετε ακριβώς την ίδια συμπεριφορά στο παιχνίδι σας. Έτσι θα έχετε την ακριβή εικόνα του τι έχετε να υλοποιήσετε.

Ειδικές οδηγίες

Δομή του terrain και όρια κίνησης χαρακτήρα



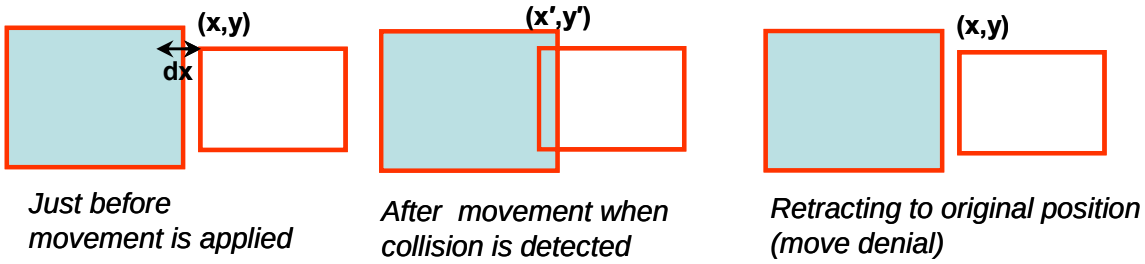


Ένα βασικό θέμα είναι η δομή του terrain ώστε να πετυχαίνεται η επιβολή της κίνησης των χαρακτήρων πάντοτε στα επιτρεπτά όρια και με σεβασμό στα αδιάβατα αντικείμενα – εμπόδια. Ο τρόπος με τον οποίο θα πετύχετε κάτι τέτοια είναι η μοντελοποίηση των εμποδίων με ορθογώνια, όπως φαίνεται στο παραπάνω σχήμα. Η υλοποίηση της μεθόδου είναι ιδιαίτερα εύκολη όπως φαίνεται και με τον παρακάτω σκελετό κώδικα.

```
struct MoveTry: public std::pair<const Sprite&, std::pair<int, int> > {
	const Sprite& sprite (void) const { return first; }
	const int&   dx (void) const     { return second.first; }
	const int&   dy (void) const     { return second.second;
	MoveTry(const Sprite& s, int _dx, int _dy):
		first(s), second.first(_dx), second.second(_dy){}
};

class Obstacle {
	unsigned x,y,w,h;
public:
	bool AllowsMove (const Sprite& s, int dx, int dy) const {
		return s.GetX() + dx + s.GetW() < x || s.GetX() + dx > x+w ||
			s.GetY() + dy + s.GetH() < y || s.GetY() + dy > y+h);
	}
	struct DenyMovePred : public std::binary_function<Obstacle,MoveTry, bool> {
		bool operator()(const Obstacle& o, const MoveTry& m) const
			{ return !o.AllowsMove(m.sprite(),m.dx(),m.dy()); }
	};
	...
};

class ObstacleHolder {     // Singleton class.
	std::list<Obstacle> obstacles;
	public:
	bool CanMove (const Sprite& s, int dx, int dy) {
		return std::find_if(
				obstacles.begin(),
				obstacles.end(),
				std::bind2nd(
					Obstacle::DenyMovePred(),
					MoveTry(s, dx, dy)
				)
			) != obstacles.end();
	}
	...
};

class PacMan : public Sprite {
public:
	void Move (int dx, int dy) {
		if (ObstacleHolder::GetSingleton().CanMove(*this, dx, dy))
			x += dx; y += dy;
	}
	...
};
```

Αυτό που γίνεται προφανές είναι ότι ο έλεγχος αυτός εφαρμόζεται για κάθε κίνηση οποιουδήποτε χαρακτήρα. Π.χ., ας θεωρήσουμε ότι έχουμε 4 ghosts και τον Pac Man που κινούνται με ταχύτητα 5 pixels ανά 100 msecs, ενώ υπάρχουν περίπου 40 εμπόδια. Αυτό σημαίνει ανά δευτερόλεπτο συνολικά: 10 x 5 x 40 = 2000 ελέγχους, ή για ένα επιθυμητό frame rate 50 fps μέσο όρο 40 έλεγχοι. Προφανώς μιλάμε για αμελητέες ποσότητες ώστε να χρειαστεί να κάνουμε βελτιστοποίηση του αλγορίθμου για obstacle collision detection.

Να σημειώσουμε ότι η μέθοδος αυτή είναι διαφορετική από το γνωστό collision detection, καθώς: (α) το τελευταίο εφαρμόζεται ***μετά*** την κίνηση ενώ το πρώτο ***πριν*** την κίνηση, και (β) το collision detection απαιτεί πραγματικά sprites και όχι απλά γεωμετρικά σχήματα. Θα μπορούσατε ωστόσο να ανάγετε το obstacle collision σε sprite collision με το να κάνετε απλώς record την προηγούμενη θέση του χαρακτήρα και όταν υπάρξει collision με ένα εμπόδιο που «αρνείται» ουσιαστικά την κίνηση να επαναφέρουμε αυτομάτως την προηγούμενες θέση (βλέπε στο παρακάτω σχήμα).



Στην περίπτωση αυτή ή χρειαζόμαστε pixel-perfect collision detection ή αλλιώς θα πρέπει να συναρμολογήσουμε τα εμπόδια από μικρότερα sprites (όπως κάναμε και πριν) για τα οποία το bounding box collision να έχει την απαραίτητη ακρίβεια (π.χ., στην περίπτωση ενός εμποδίου Τ σχήματος χρειαζόμαστε δύο τέτοια sprites).

Στη δεύτερη επιλογή θα πρέπει να ορίσουμε ένα αντίστοιχο obstacle class το οποίο θα «έχει» διάφορα συστατικά sprites ως members (δηλ. το obstacle γίνεται κάτι σαν πλατφόρμα) – θα μπορούσε να συντίθεται ένα obstacle από συστατικά ευκολότερα από ότι εάν ήταν ένα συμπαγές ολόκληρο sprite. Προφανώς θα πρέπει κάθε συστατικό sprite να έχει και το δικό του film το οποίο σημαίνει λίγη περισσότερη «φασαρία» ως προς την κατασκευή του ίδιου terrain. Ενδέχεται κάτι τέτοιο να σας φαίνεται ότι είναι κατασκευαστικά ανούσιο, ωστόσο θα δώσει επιπλέον δυνατότητες ως προς την εισαγωγή επιπλέον δράσης μέσα στο ίδιο το παιχνίδι. Π.χ.:

- Εμπόδια μπορεί να κρύβονται / επανεμφανίζονται ανάλογα με τη λογική του παιχνιδιού, όπως, π.χ., κάποιο ειδικό bonus που λαμβάνει ο παίκτης.

- Ορισμένα εμπόδια μπορεί να είναι διαβατά για τον Pac Man αλλά όχι για το Ghosts (αρκεί να μην «ενδιαφέρεται» το Pac Man sprite για collision detection ως προς αυτά τα εμπόδια).

- Μπορεί να μετακινούνται και να εφαρμόζουν κάποιο animation το οποίο μπορεί να κρύβει ή να ανοίγει κάποιους διαδρόμους (αυτό είναι πιο πολύπλοκο καθώς θέλει επιπλέον υπολογισμούς αφού ένα εμπόδιο φυσιολογικά θα πρέπει να μπορεί να σπρώχνει sprites κατά την κίνησή του).

Για να κάνουμε λοιπόν το παιχνίδι πιο ενδιαφέρον σε σχέση με την αυθεντική του έκδοση εισάγουμε κινούμενα έξυπνα εμπόδια τα οποία μπορούμε να τα υλοποιήσουμε όπως αναλύεται παρακάτω. Τα εμπόδια αυτά έχουν τη δυνατότητα όταν κινούνται να μετακινήσουν και κάποιους χαρακτήρες του παιχνιδιού.

Υλοποίηση κινούμενων έξυπνων εμποδίων

Το συστατικό στοιχείο ενός εμποδίου είναι ένα sprite το οποίο έχει δύο ειδικά χαρακτηριστικά: (α) όταν κάνει collision με ένα sprite actor του παιχνιδιού, το επαναφέρει στην αρχική θέση, και (β) όταν μετακινείται, μετακινεί αυτόματα όποιο sprite βρεθεί στο δρόμο του. Πρώτα φροντίζουμε το Sprite class να έχει τη δυνατότητα να κάνει retract στην προηγούμενη του θέση πριν καν εφαρμόσει την κίνηση. Ας ονομάσουμε αυτή την συνάρτηση *BackOff* ορίζοντας ένα game sprite class.

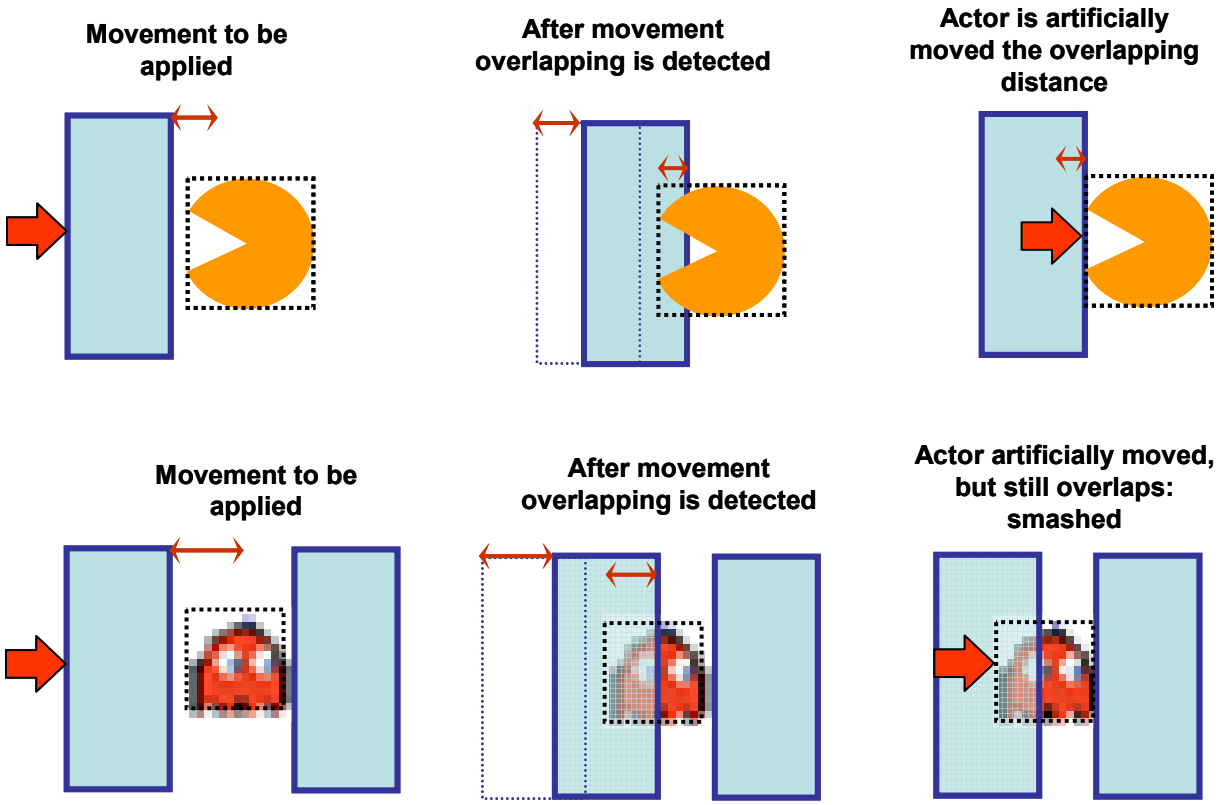
```
class GameSprite : public Sprite { // Extra stuff.
public:
	typedef void (*SmashedCallback)(GameSprite* s, void* closure);
protected:
	Dim oldX, oldY;
	SmashedCallback	onSmashed
	void*		smashedClosure;
public:
	typedef void (*SmashedCallback)(GameSprite* s, void* closure);

	void SetOnSmashed (SmashedCallback f, void* c = (void*) 0)
		{ onSmashed = f, smashedClosure = c; }

	void NotifySmashed (void)
		{ if (onSmashed) (*onSmashed)(this, smashedClosure); }

	void Move (int dx, int dy) {
		oldX = x;
		oldY = y;
		Sprite::Move();
	}
	void BackOff (void) { x = oldX; y = oldY; }
	...
};
```

Έπειτα ορίζουμε έναν νέο είδος Sprite ώστε να συμπεριφέρεται ως obstacle component, το οποίο γνωρίζει να κάνει retract όποιο sprite το ακουμπήσει, αλλά και να «σπρώχνει» όποιον βρεθεί στο δρόμο του. Για το τελευταίο και πάλι χρειαζόμαστε ειδικό registration list. Όταν ένα obstacle δεν μπορεί να σπρώξει ένα sprite σημαίνει ότι ουσιαστικά το στριμώχνει, έτσι κάνουμε post ένα smash notification.

```
class ObstacleSprite : public Sprite {
	std::list<Sprite*> pushable;
public:
	void AddPushable (Sprite* s) { pushable.push_back(s); }

	void Move (int dx, int dy) {
		for each x in pushable  {
			if (dx) {
				if after applying dx, 'this' collides with x then {
					Force x to Move() back so that it only touches 'this';
					if x still overlaps with 'this' then {
						x->NotifySmashed();
						continue;
					}
				}
			}
			if (dy) {
				IN A SIMILAR WAY AS ABOVE
			}
		}

		Sprite::Move(dx, dy);
	}
```

```
        static void WhenHit (Sprite* self, Sprite* actor, void* unused) {
                actor->BackOff();
        }

        ObstacleSprite(Dim x, Dim y, AnimationFilm* f):
                Sprite(x, y, f) { SetCollisionCallback(&WhenHit); }
};
```

Ο τρόπος με τον οποίο η κίνηση των εμποδίων επηρεάζει τους χαρακτήρες (actors, δηλ. sprites) του παιχνιδιού αποτυπώνεται στην παραπάνω εικόνα. Στην πάνω πλευρά φαίνεται μία κίνηση εμποδίου η οποία παρασύρει τον Pac Man σε ολόκληρη την απόσταση «οριζόντιας κίνησης προς τα δεξιά». Σημαντικό είναι ότι σε αυτή την περίπτωση δεν τοποθετούμε απλώς τον Pac Man στην τελική θέση, αλλά *πάντοτε καλούμε τη βασική Move()* συνάρτηση ούτως ώστε να γίνουν όλοι οι απαραίτητοι έλεγχοι για το εάν η ζητούμενη κίνηση είναι τελικά εφικτή. Το τελευταίο είναι αναγκαίο καθώς, όπως φαίνεται στο κάτω τμήματα της εικόνας, ενδέχεται αυτή η τεχνητή κίνηση λόγω σπρωξίματος του χαρακτήρα από το εμπόδιο απλώς να μην είναι δυνατή. Ειδικότερα, καθώς θα καλέσουμε τη Move() για το φάντασμα, αυτό θα κάνει collide με το εμπόδιο στα δεξιά του, του οποίου η collision function WhenHit (βλέπε παραπάνω κώδικα), θα αναγκάσει το φάντασμα να κάνει BackOff. Επομένως η κίνηση θα απαγορευθεί και θα εντοπιστεί ότι *x still overlaps with 'this'*, και σύμφωνα με τον παραπάνω κώδικα, επομένως θα κληθεί η NotifySmashed. Είναι δική σας ευθύνη να περάσετε μία κατάλληλη τέτοια callback η οποία θα πρέπει να σηματοδοτεί την «εξολόθρευση» ενός χαρακτήρα στο παιχνίδι στην περίπτωση που έχουμε smashing notification.



Η πλατφόρμα – εμπόδιο απλώς αποτελεί ένα place holder class από ObstacleSprite instances, τα οποία τοποθετούνται σε κατάλληλες σχετικές θέσεις. Ο ορισμός των εμποδίων καλό είναι να γίνεται σε ένα configuration file όπου μπορεί περιγράφεται ένα εμπόδιο ως εξής - έπεται η σκιαγράφηση ενός αντίστοιχου class:

Obstacle → *Position Total (Sprite)+*
Sprite → *Film RelativePosition AnimationType*

```
class ObstaclePlatform {
	std::list<Sprite*> sprites;
	unsigned x, y;
public:
	void SetCollisionCheck (Sprite* actor, bool flag) {
		for each x in sprites
			if (flag)
				CollisionChecker::Register(x, actor);
			else
				CollisionChecker::Cancel(x, actor);
	}
	void Add (Sprite* s, unsigned rx, unsigned ry) {
		sprites.push_back(s);
		s.SetPosition(x+rx, y+ry); // Positioned relatively to platform.
	}

	void Move (int dx, int dy) {
		For each x in sprites
			x->Move(dx, dy);
	}
	...
};
```

Πράκτορες φαντάσματα – έλεγχος από δεύτερο παίκτη

Ένα χαρακτηριστικό που θέλουμε να μπορεί να υποστηριχθεί είναι να μπορεί κάποιο από τα φαντάσματα να ελέγχεται από δεύτερο παίκτη. Όταν το παιχνίδι τρέχει σε αυτό το mode, ο δεύτερος παίκτης μπορεί να ανταγωνίζεται τον πρώτο παίκτη ακόμη και όταν δεν είναι η σειρά του. Τότε ο έλεγχος επιτρέπεται μόνο για ένα περιορισμένο χρονικό διάστημα (π.χ. 10 δευτερόλεπτα – από configuration), ενώ μόλις παρέλθει αυτό το διάστημα ο έλεγχος μεταβιβάζεται σε άλλο φάντασμα (δηλαδή γίνεται κύκλος ως προς το πιο φάντασμα ελέγχεται κάθε φορά). Αυτό θα κάνει το παιχνίδι (πιθανότατα) πιο ενδιαφέρον.

Βασικά περιεχόμενα – τελίτσες και μπόνους

Κάθε μία από τις απλές «τελείες» του παιχνιδιού, ας τις λέμε simple marks, είναι επίσης ένα sprite. Η διαφορά με τα υπόλοιπα είναι ότι ενδιαφέρεται μόνο για collision με τον Pac Man και στην περίπτωση αυτή απλώς αυτοκαταστρέφονται. Ο τρόπος με τον οποίο τοποθετούνται μέσα στο παιχνίδι βολεύει να είναι προκαθορισμένος, με τη μορφή ενός απλού «χάρτη παιχνιδιού» που δίνεται σε ένα configuration file. Προφανώς πρέπει να υπολογίσετε τις συντεταγμένες από το αυθεντικό terrain ή ένα προσχεδιασμένο δικό σας terrain και έπειτα να τις ενσωματώσετε στο αρχείο αυτό. Ένα επιπλέον χαρακτηριστικό που μπορείτε να υλοποιήσετε είναι να υπάρχουν simple marks τα οποία εκτελούν κάποιο απλό animation, π.χ. πηγαίνουν πάνω / κάτω , κυκλικά, δεξιά / αριστερά, κλπ, αλλά πάντοτε επαναφέρονται στην αρχική τους θέση. Σημαντικό είναι ότι δεν «ασχολούνται» καθόλου με τα εμπόδια ούτε τα εμπόδια με αυτά. Με τον τρόπο αυτό γίνεται κάπως δυσκολότερο το «φάγωμα» των simple marks από τον Pac Man. Οι μεγάλες τελείες με τις οποίες ο Pac Man αποκτά την προσωρινή δυνατότητα εξολόθρευσης των φαντασμάτων, θα τα λέμε power marks, καταστρέφονται επίσης κατά το collision, αλλά ενεργοποιούν τη δυνατότητα του Pac Man για χρονικό διάστημα *t*. Και τα power marks ορίζονται στο αρχείο του χάρτη του παιχνιδιού (μπορούν να υπάρχουν πολλά αρχεία ανά ξεχωριστό terrain).

Έλεγχος χρονισμού της δράσης

Για τον έλεγχο της δράσης απαιτείται προσεκτικός χρονισμός. Π.χ. για την εναλλαγή των animations στα εμπόδια καθώς και για τον έλεγχο της αντίληψης των φαντασμάτων σε τακτά χρονικά διαστήματα. Αυτό μπορεί να υλοποιηθεί εύκολα με ένα ειδικό τύπο animation και animator για τα λεγόμενα time ticks. Η υλοποίηση τους είναι πολύ απλή και βοηθούν στη χρονο-δρομολόγηση, στο ίδιο πάντα thread, διαφόρων ενεργειών. Ο ορισμός τους σκιαγραφείται παρακάτω.

```
class TickAnimation : public Animation {
public:
	typedef void (*TickFunc)(void* closure);
private:
	delay_t		delay;
	byte		repetitions;
	TickFunc	action;
	void*		closure;
public:
	TickAnimation (animid_t id) :
		Animation(id),
		delay(0), repetitions(1),
		action((TickFunc)0), closure((void*) 0){}
};

class TimerTickAnimator : public Animator {
public:
	void Progress (timestamp_t currTime);
	TimerTickAnimator (TickAnimation* tick);
};
```

Κάθε φορά που συμπληρώνεται ένα delay, εκτελείται το tick action, ενώ η διαδικασία αυτή λέγεται ένα time tick. Όταν συμπληρωθούν τόσα time ticks όσα η τιμή του repetitions, ο animator σταματάει, ενώ εάν repetitions == 0, το animation δεν σταματάει ποτέ (εκτός και εάν γίνει κλήση του Stop function).

Ομάδες εργασίας και παράδοση

Η εργασία μπορεί να περατωθεί από ομάδα τριών (3) το πολύ ατόμων. Το ποσοστό της τελικής βαθμολογίας που καταλαμβάνει η εργασίας σας είναι 40%. Η παράδοση της εργασίας ορίζεται (πιθανότατα) για την δεύτερη εβδομάδα του Ιανουαρίου, 2007, ενώ όπως συνηθίζεται θα γίνει παρουσία όλων.

Όλες οι εργασίες θα τεθούν ως διαθέσιμες στο Διαδίκτυο, τόσο σε εκτελέσιμη μορφή (ανάλογα με την πλατφόρμα) όσο και ως κώδικας (προαιρετικό), μέσα από τη σελίδα του μαθήματος, με τα στοιχεία (ονοματεπώνυμο) των δημιουργών τους.

Φροντίσετε ως αρχική στην οθόνη του παιχνιδιού να εμφανίζονται στο κέντρο της οθόνης (δηλ. με οριζόντια και κάθετη στοίχιση): τα ονόματα της ομάδας σας με λατινικούς χαρακτήρες και κεφαλαία γράμματα καθώς και η ένδειξη “University of Crete *n* Department of Computer Science *n* CSD454, Software Engineering of Intelligent, Multimedia and Mobile User Interfaces *n* Term Project, Fall Semester 2006”.