

ΑΝΑΠΤΥΞΗ ΕΞΥΠΝΩΝ ΔΙΕΠΑΦΩΝ ΚΑΙ ΠΑΙΧΝΙΔΙΩΝ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ- PROJECT

ΔΙΔΑΣΚΩΝ: Α. ΣΑΒΒΙΔΗΣ

2011-2012


ΕΠΙΜΕΛΕΙΑ ΠΡΟΕΤΟΙΜΑΣΙΑΣ:

Έφη Καρουζάκη, Νίκος Κουτσόπουλος, Πέτρος Παπανικολάου, Ιωάννης Λιλής

Σε αυτή την εργασία θα υλοποιήσετε το κλασικό παιχνίδι “Popeye”. Ένα από τα πολλά βιντεάκια στα οποία μπορείτε να δείτε το παιχνίδι είναι αυτό <http://www.youtube.com/watch?v=Qf3161wYpso>. Σας προτείνουμε όμως να ψάξετε και μόνοι σας και να παίξετε αρκετές φορές το παιχνίδι στο internet ώστε να παρατηρήσετε και και να καταλάβετε καλύτερα τους μηχανισμούς του παιχνιδιού και το βασικό gameplay.


Ζητούμενο της εργασίας θα είναι **μονάχα η πρώτη πίστα του παιχνιδιού**. Αν θέλετε όμως μπορείτε να υλοποιήσετε και τη δεύτερη πίστα σαν bonus (προαιρετικά).


Στις σελίδες που ακολουθούν θα βρείτε αναλυτικά τους βασικούς μηχανισμούς του gameplay που πρέπει να υλοποιηθούν καθώς και χρήσιμες κατασκευαστικές οδηγίες που θα σας βοηθήσουν στην υλοποίηση.


 **Ιδέα!** Στη δικιά σας υλοποίηση μπορείτε να κρατήσετε το βασικό gameplay ίδιο και να παίξετε με διάφορες άλλες παραμέτρους ώστε να δώσετε τη δική σας πινελιά στο παιχνίδι. Για παράδειγμα μπορείτε να φτιάξετε διαφορετικό terrain ή να χρησιμοποιήσετε διαφορετικά bitmaps, να βάλετε περισσότερες ή λιγότερες σκάλες στο παιχνίδι, να αλλάξετε την ταχύτητα και τη συχνότητα με την οποία πέφτουν οι καρδιές, να αλλάξετε τον αρχικό αριθμό ζώων του παίκτη, να αλλάξετε την ταχύτητα με την οποία πετάει ο Brutus τα μπουκάλια κτλ. Μπορείτε ακόμα να βάλετε cheats, ή να φτιάξετε ένα multiplayer mode όπου ένας δεύτερος παίκτης θα αναλαμβάνει την κίνηση του Brutus. Μπορείτε ακόμη να βάλετε αντικείμενα που αλλάζουν την ταχύτητα του Brutus ή απλά δίνουν πόντους στον παίκτη – γενικά μπορείτε να παίξετε και να πειραματιστείτε με ότι θέλετε, αρκεί να υλοποιήσετε τα βασικά χαρακτηριστικά του παιχνιδιού που είναι και τα ζητούμενα.


Ενότητα 1: Βασικό Gameplay

Χαρακτήρες του παιχνιδιού:

Popeye 

Olive 

Brutus 

Swee' pea  (μόνο στη δεύτερη πίστα - bonus)

1η πίστα (Υποχρεωτική)



Η Olive στην πρώτη πίστα πετάει **καρδιές**. Αυτές κάνουν κάποιο zig-zag animation και αλλάζουν και frames πέφτοντας. Όσες καρδιές φτάσουν στη θάλασσα σταματάνε την κίνησή τους και αρχίζουν να αναβοσβήνουν, σηματοδοτώντας έτσι ότι έχει μείνει λίγος χρόνος για να μαζευτούν. Αν ο Popeye δεν προλάβει μαζέψει κάποια καρδιά τότε χάνει ζωή. Κάθε καρδιά προσθέτει στον Popeye βαθμούς (score). Οι βαθμοί που δίνουν όμως εξαρτώνται από τη θέση στην οποία πιάστηκαν. Δηλαδή αν ο Popeye πιάσει μια καρδιά στο πάνω-πάνω επίπεδο (αυτό με την ταμπέλα «Thru»), αυτή δίνει 500 πόντους. Αν την πιάσει στο αμέσως αποκάτω επίπεδο (σε αυτό δηλ που βρίσκονται ο Popeye και ο Brutus στο παραπάνω screenshot) δίνει 300, στο επόμενο 100 και τέλος, αν την πιάσει όταν αυτή

είναι στο τελευταίο επίπεδο ή βρίσκεται στην επιφάνεια της θάλασσας δίνει μόνο 50 πόντους.

Ο παίκτης (Poreye) έχει σαν στόχο να μαζέψει τις καρδιές που του πετάει η Olive. Όπως είπαμε και παραπάνω πρέπει να τις μαζέψει όλες με κάποιο χρονικό περιθώριο, αλλιώς θα χάσει μια ζωή. Όταν χάσει όλες του τις ζωές τότε είναι *Game Over*. Συνολικά πρέπει να μαζευτούν τόσες καρδιές όσες οι θέσεις πάνω δεξιά (24) για να ολοκληρωθεί η πίστα.

Κίνηση του παίκτη:

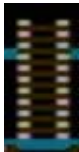
Η κίνησή του παίκτη (Poreye) μπορεί να είναι δεξιά - αριστερά πάνω στις μπλε πλατφόρμες (συμπεριλαμβανομένης της επιφάνειας της θάλασσας) και πάνω - κάτω στις σκάλες. Επίσης ο Poreye μπορεί να δώσει και μπουνιά. Όταν ο Poreye δεν πατάει σε κάποια σκάλα ή πλατφόρμα, «πέφτει» στην αποκάτω πλατφόρμα. Η κίνηση σταματάει όταν φτάσουμε στις άκρες του terrain (πίσω από τις σκάλες), εκτός αν βρισκόμαστε στην πάνω-πάνω πλατφόρμα που έχει την πινακίδα «Thru». Σε αυτή την πλατφόρμα, όταν ο Poreye φτάσει στην άκρη του terrain (και το sprite τους βγει εξ ολοκλήρου έξω από το οπτικό μας πεδίο) από τη μια πλευρά, τότε εμφανίζεται (βγαίνει) από την άλλη μεριά, σαν να ήταν δηλαδή κυκλικό το terrain σε αυτό το σημείο.

(*) Όταν ο Poreye φτάσει σε μια σκάλα, πρέπει να επιλέξει να κατέβει ή να ανέβει αντίστοιχα, ενώ όταν φτάσει στην άκρη της πλατφόρμας πέφτει αυτόματα.

Οι σκάλες:

Στο παιχνίδι συναντάμε 2 ειδών σκάλες:

Σκάλα 1:



Σε αυτή τη σκάλα ο παίκτης μπορεί να ανέβει ή να κατέβει **σε μια κίνηση**. Δηλαδή όταν ο παίκτης βρίσκεται στην κορυφή της σκάλας και επιλέξει να κινηθεί προς τα κάτω, τότε ο Poreye θα αρχίσει να την κατεβαίνει και η κίνησή του θα σταματήσει μόνο όταν βρεθεί στη βάση της σκάλας. Αντίστοιχα ισχύουν και για να την ανέβει.

Σκάλα 2:



Σε αυτή τη σκάλα η κίνηση γίνεται **ανά σκαλοπάτι**. Δηλαδή όταν ο παίκτης βρίσκεται στην κορυφή της σκάλας και επιλέξει να κινηθεί προς τα κάτω τότε ο Poreye θα κατέβει μονάχα ένα σκαλί και η κίνησή του θα σταματήσει εκεί.


Αντίστοιχα θα γίνει η κίνηση και στο ανέβασμα της σκάλας.

Προσέξτε ότι το animation που έχει ο Poreye για να ανέβει την κάθε σκάλα διαφέρει. Δηλαδή και το bitmap αλλάζει, και η κίνηση είναι διαφορετική (για να ανέβει τη σκάλα 1 χρειάζεται να αλλάξει μόνο το y coordinate του Poreye ενώ για να ανέβει τη σκάλα 2 αλλάζει και το x και το y).


Η μπουνιά:



Ο Poreye χρησιμοποιώντας τη μπουνιά μπορεί να κάνει τα παρακάτω:

1. Μπορεί να «σπάσει» τα αντικείμενα που του πετάει ο Brutus.
2. Μπορεί να φάει το **σπανάκι** που θα βρίσκεται σε μια από τις 2 θέσεις (τις έχουμε σημειώσει στην εικόνα με ). Όταν συμβεί αυτό, η μουσική αλλάζει, ο Poreye γίνεται ροζ για λίγα δευτερόλεπτα και γίνεται «άτρωτος». Σε αυτή την κατάσταση:

- i. Ο Poreye μπορεί να κυνηγήσει τον Brutus και να τον χτυπήσει (δε χρειάζεται να δώσει μπουνιά, αρκεί να τον ακουμπήσει). Όταν χτυπηθεί ο Brutus, κάνει κάποιο animation και πέφτει στη θάλασσα, ενώ ο Poreye κερδίζει βαθμούς (score).
- ii. Οι καρδιές **παγώνουν** στις θέσεις που βρίσκονται.
- iii. Οι βαθμοί που δίνουν οι καρδιές **διπλασιάζονται** (οπότε πάλι ανάλογα με το επίπεδο δίνουν 1000, 600, 200 και 100 από πάνω προς τα κάτω).

Το σπανάκι **δεν** μένει στην ίδια θέση συνεχώς αλλά αλλάζει κάθε λίγα λεπτά (εναλλάσσεται ανάμεσα στις 2 θέσεις που έχουμε μαρκάρει με  στην παραπάνω εικόνα. Επίσης, **δεν** ανανεώνεται όταν φαγωθεί – δηλαδή ο Poreye μπορεί να το χρησιμοποιήσει μόνο μια φορά. Το σπανάκι ξαναβγαίνει άμα ο παίκτης χάσει ζωή ή άμα αλλάξει πίστα.

3. Μπορεί να χτυπήσει το μποξ πέφτοντας από την πάνω αριστερή μπλε πλατφόρμα (αυτή με την ταμπέλα «Thru»). Όταν το χτυπήσει, εκείνο κάνει ένα animation και «χτυπάει» τον κουβά. Εκείνος με τη σειρά του πέφτει. Αν ο Brutus βρεθεί στην πορεία του κουβά τότε τον καπακώνει και τον ακινητοποιεί για λίγο, και ο Poreye παίρνει πάλι βαθμούς (bonus). Ο κουβάς δεν ανανεώνεται και ως εκ τούτου μπορεί να χρησιμοποιηθεί μόνο μια φορά (ξαναβγαίνει μόνο όταν ο παίκτης χάσει ζωή).



Ο Brutus περιφέρεται στην πίστα με σκοπό να «πιάσει» τον Poreye. Η κίνησή του δεν είναι τυχαία αλλά έχει κάποιο στοιχειώδες AI (συνήθως πάει προς τον Poreye). Όταν ο Poreye είναι στην ίδια πλατφόρμα με τον Brutus, και η απόστασή τους είναι σχετικά μικρή, ο Brutus πετάει αντικείμενα (μπουκάλια?) στον Poreye για να τον χτυπήσει. Όταν ο Poreye φάει σπανάκι, τότε το AI του Brutus αλλάζει λογική και προσπαθεί να τον αποφύγει.

Ο Brutus διαθέτει κινήσεις που του επιτρέπουν να φτάσει τον Poreye ακόμα και αν αυτός δεν βρίσκεται στην ίδια πλατφόρμα, αλλά βρίσκεται στην αμέσως αποπάνω ή αποκάτω. Διαθέτει επίσης μια κίνηση με την οποία πηδάει απευθείας στην αποκάτω πλατφόρμα, προκαλώντας παράλληλα ένα μικρό σεισμό (earthquake effect) στο terrain. Όταν ο Brutus φτάσει τον Poreye με οποιοδήποτε τρόπο (είτε από την ίδια πλατφόρμα είτε από την αποπάνω /αποκάτω) είτε καταφέρει να τον χτυπήσει με κάποιο από τα αντικείμενα που του πετάει, τότε ο τελευταίος χάνει μια ζωή.

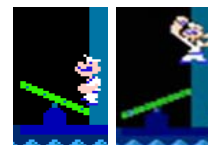


Η Olive σε αυτήν την πίστα πετάει νότες τις οποίες κατ' αντιστοιχία με τις καρδούλες πρέπει να τις μαζεύει ανελλιπώς ο Poreye. Οι νότες τώρα δεν μπαίνουν στην πάνω αριστερή γωνία της οθόνης αλλά η καθεμία έχει τη θέση της, και φαίνεται να «γεμίζουν». Και εδώ ισχύει ότι όσες φτάσουν στη θάλασσα αρχίζουν να αναβοσβήνουν και ο Poreye έχει λίγο χρόνο ακόμα για να τις μαζέψει. Συνολικά πρέπει να μαζευτούν 16 νότες (όσες και οι θέσεις δηλαδή) για να ολοκληρωθεί η πίστα.

Ο Poreye κινείται με τον ίδιο τρόπο που κινείται και στην πρώτη πίστα. Και εδώ υπάρχει μια πλατφόρμα με την πινακίδα «Thru». Κι εδώ υπάρχει σπανάκι, που όμως δεν αλλάζει θέσεις – είναι σταθερά στο σημείο πίσω από τη σκάλα της μεσαίας πλατφόρμας. Μπορεί και πάλι να καταναλωθεί μόνο μια φορά. Επιπλέον ο Poreye μπορεί να χρησιμοποιήσει την τραμπάλα όπως περιγράφεται παρακάτω.

Ο Brutus έχει τις ίδιες κινήσεις και το ίδιο AI με την πρώτη πίστα. Επιπλέον μπορεί να χρησιμοποιήσει την τραμπάλα όπως περιγράφεται παρακάτω.

Η τραμπάλα. Σε αυτήν την πίστα δεν υπάρχει μποξ και βαρίδι. Υπάρχει όμως μια τραμπάλα. Όταν κάποιος χαρακτήρας πέσει πάνω στην τραμπάλα από την τελευταία πλατφόρμα (πέφτει πάντα στη δεξιά πλευρά της τραμπάλας που είναι και η σηκωμένη), η τραμπάλα κάνει ένα animation και «πετάει» τον χαρακτήρα ψηλά. Τον Poreye τον πετάει πάνω τόσο ώστε να χτυπήσει τον swee' pea. Ο τελευταίος όταν χτυπηθεί δίνει λεφτά (δηλαδή score). Όταν ο Brutus πέσει στην τραμπάλα, τον πετάει απλά στην αποπάνω πλατφόρμα.



Terrain

Το terrain του παιχνιδιού είναι χωρισμένο σε 5 επίπεδα όπως φαίνεται και στην Εικόνα 1. Στο υψηλότερο επίπεδο (επίπεδο 1) βρίσκεται πάντα μόνο η Olive. Τα υπόλοιπα επίπεδα μπορούν να περιέχουν πλατφόρμες, οι οποίες χρησιμοποιούνται σα δάπεδο για τον Brutus και τον Poreye και οι οποίες μπορούν να έχουν διάφορα μεγέθη. Έτσι ένα επίπεδο μπορεί να περιέχει πολλές πλατφόρμες που αφήνουν κενό μεταξύ τους (όπως στο επίπεδο 2) ή να έχει μια μεγάλη πλατφόρμα που να πιάνει όλο το «δάπεδο» του επιπέδου (όπως είναι τα επίπεδα 3 και 4). Το επίπεδο 5 είναι στην επιφάνεια της θάλασσας και έχει πάντα μια μεγάλη πλατφόρμα που δεν αφήνει κενά.



Εικόνα 1: Terrain παιχνιδιού για την πρώτη πίστα.



Εικόνα 2: Κάθε επίπεδο μπορεί να έχει πλατφόρμες σαν "πάτωμα" οι οποίες μπορούν να αφήνουν κενά μεταξύ τους. Στο Επίπεδο 5 δεν επιτρέπονται κενά. Στο Επίπεδο 1 δεν έχουμε πλατφόρμες για τον παίκτη - εκεί βρίσκεται μόνο η Olive.

Έλεγχος χρονισμού της δράσης

Για τον έλεγχο της δράσης απαιτείται προσεκτικός χρονισμός σε αρκετά σημεία του παιχνιδιού. Αυτό μπορεί να υλοποιηθεί εύκολα με ένα ειδικό τύπο animation και animator, τα λεγόμενα **time ticks** (*TimeTickAnimation*, *TimeTickAnimator*). 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).

Έλεγχος κίνησης του χαρακτήρα

Οι χαρακτήρες του παιχνιδιού κινούνται πάνω σε πλατφόρμες και σε σκάλες.

Πλατφόρμες

Όταν οι χαρακτήρες σε κάποια κίνηση βρεθούν χωρίς να πατάνε σε κάποια πλατφόρμα ή σε κάποια σκάλα, τότε πρέπει να πέσουν. Ένας τρόπος να το υλοποιήσετε σκιαγραφείται παρακάτω:


```

class Platform;

class PlatformHolder {    //Singleton
public:
    Platform* CheckPlatformCollision(Sprite* s) const;
};

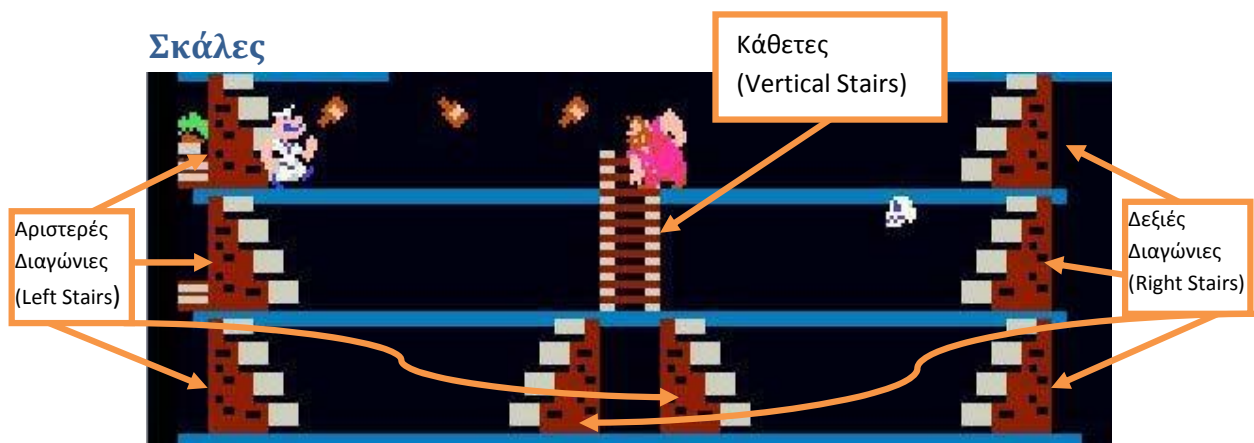
class Sprite {
public:
    virtual void Move(int dx, int dy);
};

class Popeye : public Sprite {
    Platform* platform;
    bool isFalling;
public:
    //Horizontal movement
    void MoveHorizontal(int dx) {
        //Check for obstacle collision (stage boundaries)
        //if (!isFalling)
        //    ApplyMovement(dx);
        //if (!OnPlatform())
        //    StartFalling();
    }

    virtual void Move(int dx, int dy) {
        if (isFalling) {
            assert(dx == 0);
            Platform *p = PlatformHolder::GetSingleton().
                CheckPlatformCollision(this);
            if (p){
                this->platform = p;
                //Align player with platform
            }
        }
        Sprite::Move(dx, dy);
    }
    //...
};

```

Σκάλες



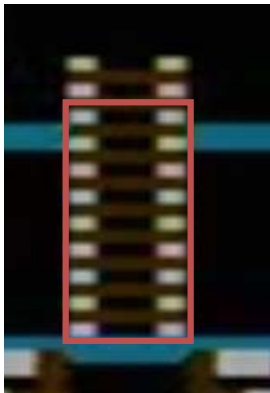
Έχουμε 2 τύπους σκάλας. Τις κάθετες και τις διαγώνιες. Οι διαγώνιες με τη σειρά τους χωρίζονται σε αριστερές και σε δεξιές. Η κίνηση του Poreye είναι διαφορετική για τους δυο τύπους.

Κάθετες σκάλες (Στο κλασσικό παιχνίδι είναι μόνο 1 και τη χρησιμοποιεί μόνο ο Poreye, αλλά η δικιά σας έκδοση μπορεί αν θέλετε να διαφέρει):

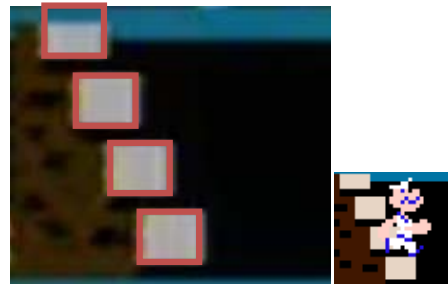
- Οι χαρακτήρες τις ανεβαίνουν σε μια κίνηση.
- Για να ανέβει μια κάθετη σκάλα, το sprite του χαρακτήρα γίνεται align στη μέση της σκάλας και μετά γίνεται ένα animation (ο χαρακτήρας γυρίζει πλάτη) και την ανεβαίνει.
- Μοντελοποιούμε την κάθετη σκάλα με ένα και μόνο rectangle.

Διαγώνιες σκάλες

- Οι χαρακτήρες τις ανεβαίνει σκαλοπάτι-σκαλοπάτι.
- Για να ανέβει μια διαγώνια σκάλα, ο χαρακτήρας “κοιτάει” προς το μέρος της σκάλας (άρα χρειάζεται άλλο animation για τις δεξιές και άλλο για τις αριστερές)
- Είναι χωρισμένες σε σκαλιά, και το κάθε σκαλί έχει το δικό του Rectangle.



Εικόνα 3: κάθετη σκάλα



Διαγώνια αριστερή σκάλα

Οι χαρακτήρες, αν δεν πέφτουν μπορεί να πατάνε είτε σε κάποια πλατφόρμα είτε σε κάποια σκάλα (και μάλιστα σε συγκεκριμένο σκαλί). Οπότε αυτή η πληροφορία πρέπει να είναι διαθέσιμη και να περιέχεται στην κλάση του χαρακτήρα. Όταν ο χαρακτήρας ζητήσει να κινηθεί για παράδειγμα «πάνω» πρέπει να ελέγξουμε αν μεν βρίσκεται σε πλατφόρμα αν είναι σε σημείο που υπάρχει σκάλα και μπορεί να ανέβει, ενώ αν βρίσκεται σε σκαλί πρέπει να ελέγξουμε αν έχει άλλα σκαλιά να ανέβει, αν πάει σε άλλη πλατφόρμα, τι κίνηση κάνει κτλ. Επειδή η κίνηση (το animation που θα κάνει ο χαρακτήρας) εξαρτάται από τον τύπο της σκάλας, καλό είναι αυτή η πληροφορία να περιέχεται στην κλάση της σκάλας. Επίσης για να ξέρουμε πότε όταν είμαστε πάνω σε μια πλατφόρμα επιτρέπεται να κινηθούμε «πάνω» ή «κάτω» θα πρέπει κάπως να ενσωματώσουμε στο platform την έννοια του ποιες σκάλες ακουμπάνε σε αυτό και πάνε προς τα πάνω ή προς τα κάτω. Για να αναπαραστήσετε τις σκάλες, μια προτεινόμενη δομή δίνεται παρακάτω:

```

struct Rect { int x, y, w, h; };

class Platform {
    typedef std::list<Stair*> StairList;
    StairList goingUp, goingDown;
    Rect rect;
public:
    Stair* CollidesWithGoingUpStair(Sprite* s) const {
        //check bottop steps of goingUp stairs
    }
    Stair* CollidesWithGoingDownStair(Sprite* s) const {
        //check top steps of goingDown stairs
    }
    //...
};

class Step {
    Rect rect;
    //...
}

class Stair {
protected:
    std::vector<Step*> steps;
    Platform *bottom, *top;
    MovingPathAnimation *up, *down;
    MovingPathAnimator *animator;

public:
    virtual void AlignUp (Sprite* s) const = 0;
    virtual void AlignDown (Sprite* s) const = 0;

    virtual void AnimateUp (Sprite* s) const = 0;
    virtual void AnimateDown(Sprite* s) const = 0;
};

```

Οι πλατφόρμες ξέρουν ποιες σκάλες ακουμπάνε σε αυτές – ποιες πάνε κάτω και ποιες πάνω.

Οι σκάλες ξέρουν σε ποιες πλατφόρμες ακουμπάνε από πάνω και από κάτω. Επίσης ξέρουν τα σκαλιά που έχουν και τι animation πρέπει να γίνει (η κάθε σκάλα ξέρει το δικό της) για να ανέβουν ή να κατέβουν οι χαρακτήρες.

Κάθε φορά που ολοκληρώνουμε το animation για να ανέβουμε ένα σκαλί, πρέπει να κάνουμε update τη θέση του χαρακτήρα (να δούμε αν τώρα που ανέβηκε βρέθηκε σε κάποια πλατφόρμα ή σε κάποιο άλλο σκαλί).

```

class DiagonalStair : public Stair {
    typedef std::pair<Sprite*, Stair*> AnimatorData;

    static void UpStepCompleted (Animator*, void* closure) {
        AnimatorData* data = (AnimatorData*) closure;
        Sprite* player = data->first;
        Stair* stair = data->second;

        if (Step* step = player->GetStep()) { //player was already on the stair
            Step* up = stair->GetUpStep(step);
            assert(up);

            if (up == stair->GetTopStep()) {
                up = (Step *) 0; //not on the stair anymore, but on the platform
                player->SetPlatform(stair->GetTopPlatform());
            }
            player->SetStep(up);
        }
        else //player was on bottom platform
            player->SetStep(stair->GetBottomStep());
        player->EnableMovement();
        delete data;
    }
    static void DownStepCompleted (Animator*, void* closure); //similar implementation

public:
    virtual void AnimateUp (Sprite* s) const {
        s->DisableMovement(); //player should not be allowed to move during the animation
        animator->SetAnimation(up);
        animator->SetOnFinish(UpStepCompleted, new AnimatorData(s, this));
        animator->Start(s);
    }
    virtual void AnimateDown (Sprite* s) const; //similar implementation
};

```

```

class LeftStair : public DiagonalStair {
    virtual void AlignUp      (Sprite* s) const {
        //Place sprite at the right of the bottom step, facing left
    }
    virtual void AlignDown    (Sprite* s) const {
        //Place sprite at the middle of the top step, facing right
    }
};

//Similar for class RightStair

class VerticalStair : public Stair {
    typedef std::pair<Sprite*, Stair*> AnimatorData;

    static void UpStepCompleted (Animator*, void* closure) {
        AnimatorData* data = (AnimatorData*) closure;
        data->first->SetPlatform(data->second->GetTopPlatform());
        player->EnableMovement();
        delete data;
    }

public:
    virtual void AlignUp      (Sprite* s) const {
        //Place sprite at the middle
    }
    virtual void AlignDown    (Sprite* s) const {
        //Place sprite at the middle
    }

    virtual void AnimateUp    (Sprite* s) const {
        s->DisableMovement();//player should not be allowed to move during the animation
        animator->SetAnimation(up);
        animator->SetOnFinish(UpStepCompleted, new AnimatorData(s, this));
        animator->Start(s);
    }
    virtual void AnimateDown (Sprite* s) const;    //similar implementation
};

```

```

class Popeye : public Sprite {
    Platform*   platform;
    Stair*      stair;
    Step*       step;
    bool        canMove;
    bool        isFalling;
public:
    //Vertical movement
    void MoveUp(void) {
        if (canMove)
            if (platform) {
                if (stair = platform->CollidesWithGoingUpStair(this)) {
                    platform = (Platform *) 0;
                    stair->AlignUp(this);
                    stair->AnimateUp(this);
                }
            }
            else if (stair)
                stair->AnimateUp(this);
    }
    void MoveDown(void);    //similar implementation

    void EnableMovement  (void) { canMove = true; }
    void DisableMovement (void) { canMove = false; }
    //...
};

```

Προσαρμογές (configuration facilities)

Μπορείτε να έχετε έλεγχο ορισμένων χαρακτηριστικών του παιχνιδιού μέσω configuration files. Ορισμένα από τα επιθυμητά configurations μέσω των οποίων θα έχετε τη δυνατότητα να κάνετε το παιχνίδι σας να μοιάζει κάθε φορά και διαφορετικό είναι τα εξής:

- Ρυθμός ρίψης των καρδίων που ρίχνονται από την από την Olive
- Ταχύτητα κίνησης των καρδίων που ρίχνονται από την από την Olive
- Ταχύτητα κίνησης των μπουκαλιών που εκτοξεύει ο Brutus
- Ο χρόνος που είναι ενεργές οι καρδιές όσο βρίσκονται στο χαμηλότερο επίπεδο
- Ο χρόνος που μπορεί να είναι άτρωτος ο Poreye
- Συχνότητα με την οποία θα αλλάζει θέση το σπανάκι
- Τις δυνατές θέσεις που θα μπορεί να πάει το σπανάκι
- Τα rectangles που αναφέρονται στις σκάλες
- Αρχικές ζωές του παίκτη
- Οτιδήποτε Άλλο.

Pause – Resume

Στο παιχνίδι σας είναι υποχρεωτικό να υλοποιηθεί ο μηχανισμός του pause και του resume. Κάθε φορά που ο παίκτης επιλέγει να «παγώσει» το παιχνίδι, θα πρέπει να εμφανίζεται μια οθόνη που θα ενημερώνει των χρήστη ότι το παιχνίδι έχει σταματήσει προσωρινά.

Ομαδική εργασία και παράδοση

Η εργασία μπορεί να περατωθεί από ομάδα τριών (3) το πολύ ατόμων. Το ποσοστό της τελικής βαθμολογίας που καταλαμβάνει η εργασία σας είναι 35% και σε εξαιρετικές περιπτώσεις πολύ καλών αποτελεσμάτων μπορεί να γίνει και 40%. Η ημερομηνία παράδοσης/εξέτασης του project θα είναι την περίοδο που μεσολαβεί μετά την λήξη της εξεταστικής και την έναρξης του νέου εξαμήνου (5/02 - 13/02).

Η παράδοση / εξέταση της όπως συνηθίζεται θα γίνει παρουσία όλων από τον διδάσκοντα (μπορείτε να φέρετε και invited άτομα αρκεί να έχουν δηλωθεί πιο νωρίς ώστε αν χρειαστεί να προγραμματίσουμε άλλο χώρο εξέτασης).

Φροντίστε η αρχική οθόνη του παιχνιδιού να περιέχει: τα ονόματα της ομάδας σας με λατινικούς χαρακτήρες και κεφάλαια γράμματα καθώς και την ένδειξη “University of Crete \n Department of Computer Science \n CS-454. Development of Intelligent Interfaces and Games \n Term Project, Fall Semester 2011”.