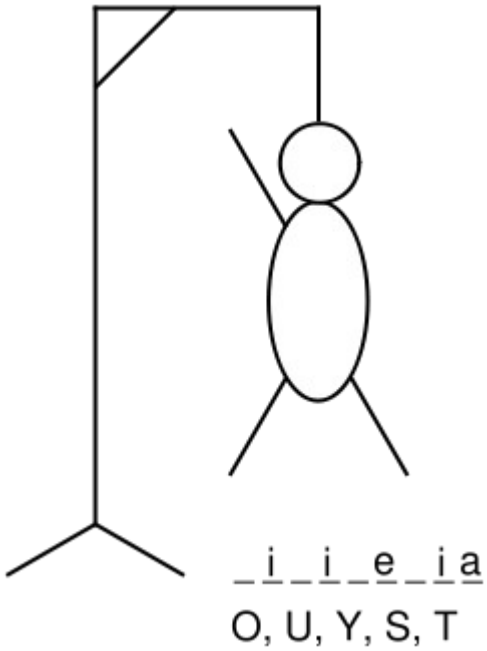
ΚΡΕΜΑΛΑ



Project 7

Εκφώνηση άσκησης

Υλοποίηση του παιχνιδιού «Κρεμάλα»

Το Project αυτό συνίσταται στην κατασκευή ενός προγράμματος σεναρίου κελύφους, που θα υλοποιεί το γνωστό παιχνίδι «Κρεμάλα» με δυνατότητα τυχαίας επιλογής λέξης από αρχείο λέξεων.

Η Κρεμάλα σαν παιχνίδι

Η κρεμάλα είναι ένα παιχνίδι μεταξύ δύο παιχτών που παίζεται με χαρτί και μολύβι. Ο ένας παίχτης σκέφτεται μια λέξη και ο άλλος προσπαθεί να την βρει μαντεύοντας τα γράμματα της ή και απευθείας την λέξη. Η μυστική λέξη που έχει σκεφτεί ο πρώτος παίχτης, αντιπροσωπεύεται από μια σειρά από παύλες φανερώνοντας τον αριθμό των γραμμάτων. Εάν ένας παίχτης μαντέψει σωστά ένα γράμμα, τότε αυτό γράφεται σε όλες τις σωστές του θέσεις αντικαθιστώντας τις παύλες. Εάν το γράμμα που πρότεινε ο παίχτης δεν υπάρχει στην λέξη, τότε ο άλλος παίχτης σχεδιάζει και ένα κομμάτι του σκίτσου ενός κρεμασμένου ανθρώπου.

Το παιχνίδι τελειώνει όταν:

- Ο παίχτης μαντέψει όλα τα γράμματα της κρυφής λέξης ή απευθείας την λέξη
- Ο άλλος παίχτης συμπληρώσει το σκίτσο.

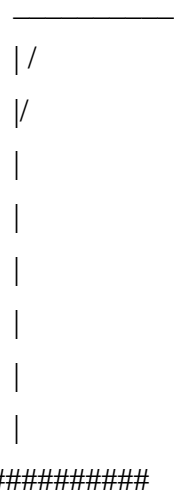
Ανάλυση

Μέρος Α – Αναφορά, ανάλυση τακτικής

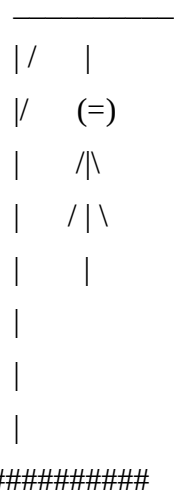
Στο πρώτο μέρος της ανάλυσης θα αναλύσουμε την εκφώνηση του προγράμματος που μας έχει ζητηθεί να αναπτύξουμε, θα καθορίζουμε τα ζητούμενα του και θα σχεδιάζουμε έναν πρόχειρο αλγόριθμο υλοποίησής του προγράμματος.

Η εκφώνηση μας ζήτα να αναπτύξουμε το παιχνίδι κρεμάλα στο περιβάλλον κελύφους του λειτουργικού συστήματος Linux. Αμέσως ένα πρόβλημα που προκύπτει είναι η αναπαράσταση του σκίτσου της κρεμάλας καθώς το κέλυφος δεν υποστηρίζει εικόνες καθαρά με την έννοια της εικόνας. Υποστηρίζει όμως χαρακτήρες, σύμβολα και αριθμούς. Έτσι θα σχεδιάσουμε το σκίτσο της κρεμάλας με χαρακτήρες. Όπως γνωρίζουμε, σε κάθε λανθασμένη εισαγωγή χαρακτήρα για την ανεύρεση της μυστικής λέξης, θα πρέπει να συμπληρώνεται και ένα σχέδιο, μέχρις ότου να ολοκληρωθεί το σκίτσο. Θα πούμε λοιπόν ότι ο χρήστης θα έχει δικαίωμα να εισάγει μέχρι και δέκα λανθασμένους χαρακτήρες. Έτσι θα χωρίσουμε και το σκίτσο μας σε δέκα διαφορετικά κομμάτια τα οποία θα αποθηκεύσουμε

αντίστοιχα σε δέκα διαφορετικά αρχεία κειμένου. Κάθε φορά που ο χρήστης θα εισάγει λάθος χαρακτήρα, θα εμφανίζεται στην οθόνη και η αντίστοιχη εικόνα. Για παράδειγμα, αν εισάγει για πέμπτη φορά έναν λάθος χαρακτήρα θα εμφανιστεί το παρακάτω σκίτσο:



αν εισάγει για όγδοη φορά έναν λάθος χαρακτήρα θα εμφανιστεί το παρακάτω:



Αυτές οι εικόνες είναι αποθηκευμένες σε μορφή κειμένου στον φάκελο “images”.

Στην κρεμάλα μας η κρυφή λέξη θα επιλέγεται τυχαία από ένα λεξικό, δηλαδή ένα αρχείο με διαθέσιμες για την κρεμάλα, λέξεις. Εκτελώντας το πρόγραμμα θα μετράμε το πλήθος των λέξεων και θα παράγουμε έναν τυχαίο αριθμό μικρότερο πάντα από το πλήθος αυτό. Ο αριθμός θα αντιπροσωπεύει την θέση στην οποία βρίσκεται η λέξη που θα επιλεχθεί από το λεξικό. Στη συνέχεια το πρόγραμμα θα μας ζητήσει να εισάγουμε έναν χαρακτήρα προς ανεύρεση της μυστικής λέξης. Μια δομή επανάληψης θα ελέγχει αν ο χαρακτήρας υπάρχει στη μυστική λέξη και αναλόγως ή θα τον εμφανίζει στην οθόνη στη θέση στην οποία θα πρέπει να είναι μέσα στη λέξη, ή θα σχεδιάσει και ένα

κομμάτι του σκίτσου. Κάθε λάθος εισαγωγή χαρακτήρα θα αναγράφεται και στην οθόνη για αποφυγή μελλοντικής εισαγωγής του ίδιου χαρακτήρα.

Η παραπάνω εκτέλεση του κώδικα θα γίνεται μέσα σε μια δομή επανάληψης η οποία θα εκτελείτε το πολύ δέκα φορές, όσες και οι διαθέσιμες λανθασμένες εισαγωγές χαρακτήρα. Στην περίπτωση όπου ο χρήστης μαντέψει την μυστική λέξη πριν το πέρας των επαναλήψεων η ροή του προγράμματος θα σταματά και αμέσως το πρόγραμμα θα τερματίζεται με αντίστοιχο συγχαρητήριο μήνυμα. Αν ο χρήστης δεν καταφέρει να μαντέψει την λέξη μετά το πέρας των δέκα επαναλήψεων, το πρόγραμμα θα τερματιστεί με αντίστοιχο μήνυμα το οποίο θα αναγράφει την μυστική λέξη και ότι ο χρήστης έχασε.

Πριν προχωρήσουμε στην υλοποίηση του προγράμματος πρέπει να προσέξουμε ότι υπάρχει περίπτωση ο χρήστης να μην εισάγει απλά έναν χαρακτήρα αλλά ένα αλφαριθμητικό Έτσι θα δημιουργήσουμε μια δομή επανάληψης μέσα στην οποία θα γίνεται η εισαγωγή του χαρακτήρα. Η δομή θα επαναλαμβάνεται μέχρις ότου ο χρήστης να εισάγει έναν χαρακτήρα και μόνο.

Ας προχωρήσουμε στην υλοποίηση ενός πρόχειρου ψευδοκώδικα που θα περιγράφει τις παραπάνω λειτουργίες.

```
ΠΡΟΓΡΑΜΜΑ ΚΡΕΜΑΛΑ

φακελος_εικονων=./images
αρχειο_λεξεων=./words
random=τυχαιος_αριθμος
λεξη=τυχαια_λεξη_απο_αρχειο

ΟΣΟ ( λανθασμενες_επιλογες < 10 ) ΕΠΑΝΕΛΑΒΕ

    ΕΜΦΑΝΙΣΕ εικονα
    ΔΙΑΒΑΣΕ χαρακτηρα

    ΟΣΟ ( ο χαρακτηρας εισαγωγης δεν είναι χαρακτηρας ) ΕΠΑΝΕΛΑΒΕ
        ΕΜΦΑΝΙΣΕ “Λαθος εισαγωγή”
        ΔΙΑΒΑΣΕ χαρακτηρα
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

    ΓΙΑ κάθε ψηφίο της λέξης ΕΠΑΝΕΑΒΕ
        ΑΝ ( ο χαρακτήρας ισουται με χαρακτηρα της λεξης ) ΤΟΤΕ
                ΕΜΦΑΝΙΣΕ τον χαρακτήρα στην θέση του στην λέξη
        ΤΕΛΟΣ_ΑΝ
```

ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΑΝ (ο χρήστης βρηκε την λέξη) ΤΟΤΕ

ΕΜΦΑΝΙΣΕ “ΝΙΚΗΣΕΣ”

ΤΕΡΜΑΤΙΣΕ

ΤΕΛΟΣ_ΑΝ

ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΑΝ (ο χρηστης δεν βρηκε την λεξη) ΤΟΤΕ

ΕΜΦΑΝΙΣΕ “ΕΧΑΣΕΣ”

ΤΕΛΟΣ_ΑΝ

ΤΕΛΟΣ_ΚΡΕΜΑΛΑ

Μέρος Β – Ανάλυση του κώδικα

Στο δεύτερο μέρος της ανάλυσης, θα αναλύσουμε τον κώδικα του προγράμματος μας. Για καλύτερη κατανόηση θα σπάσουμε τον κώδικα σε μικρά κομμάτια τα οποία θα αναλύσουμε και σχολιάσουμε.

Κομμάτι 1

```
#!/bin/bash

WORDS=./words
IMAGES=./images

if [ ! -r ${WORDS} ]; then
   echo "${WORDS} does not exist or is not readable" && exit 1
fi
```

Στην αρχή του κώδικα μας καθορίζουμε την διαδρομή στην οποία βρίσκονται οι εικόνες μας, καθώς και το όνομα του λεξικού στο οποίο περιέχονται οι υποψήφιες μυστικές λέξεις. Στην συνέχεια ελέγχουμε αν όντως το αρχείο που δώσαμε υπάρχει και αναλόγως ή συνεχίζεται η εκτέλεση του προγράμματος, ή τερματίζεται με κατάλληλο μήνυμα.

Κομμάτι 2

```
WORDSLN=$( wc -l ${WORDS} | awk '{ print $1}' )

RANDOMN=$( awk -v l=$WORDSLN 'BEGIN { srand();
                        r = rand() * l;
                        printf( "%d", ( r + 1 ) ) }' )

WORD=$( awk -v l=$RANDOMN 'BEGIN { RS="";
                       FS="\n" } { print $l }' $WORDS)

WORD_LENGTH=${#WORD}
```

Στη συνέχεια μετράμε το πλήθος των λέξεων οι οποίες βρίσκονται στο λεξικό και δημιουργούμαι έναν τυχαίο αριθμό, μικρότερο πάντα από το πλήθος των λέξεων, τον οποίο και αποθηκεύουμε στην μεταβλητή RANDOMN. Έπειτα επιλέγουμε την μυστική μας λέξη απο το λεξικό με βάση τον τυχαίο αριθμό και την αποθηκεύουμε στην μεταβλητή WORD. Τέλος στην μεταβλητή WORD_LENGTH αποθηκεύουμε το μήκος της λέξης.

Κομμάτι 3

```
for i in $( seq 0 $(( $WORD_LENGTH )) )
do
   CHAR_PIN[$i]=$(echo "${WORD}" | cut -c$(($i+1)) )

done
```

Εδώ αποθηκεύουμε τους χαρακτήρες της μυστικής λέξης έναν προς έναν στον πίνακα CHAR_PIN.

Κομμάτι 4

```
i=0
while [ $i -lt $WORD_LENGTH ]; do
        STR_GUESSES[$i]="_"
        let "i++"
done
```

Στη συνέχεια δημιουργούμε έναν πίνακα ονόματι STR_GUESSES μεγέθους ίδιου με το μήκος της μυστικής λέξης και τον γεμίζουμε με κάτω παύλες (_). Είναι ο πίνακας ο οποίος θα εμφανίζεται και

κατά την εκτέλεση του προγράμματος στην θέση της μυστικής λέξης.

Κομμάτι 5

```
GUESSED=""
flag=0
GUESSES=10

while [ $GUESSES -gt 0 ] ; do
   clear
   cat ${IMAGES}/${GUESSES}
   echo -e "\n\nYou have ${GUESSES} chance(s) remaining!"
         echo -e "\nYou have guessed so far:\n\t $GUESSED"
echo -e "\n${STR_GUESSES[@]}"
echo -e "\n\nEnter a character"
read char
```

Αφού αρχικοποιήσουν κάποιες μεταβλητές που θα μας χρειαστούν στην συνέχεια, ανοίγουμε τον πρώτο βρόγχο επανάληψης ο οποίο μετράει τις λανθασμένες εισαγωγές χαρακτήρα. Έπειτα καθαρίζουμε την οθόνη και εμφανίζουμε την πρώτη μας εικόνα, πόσες λανθασμένες εισαγωγές έχει το δικαίωμα να κάνει ο χρήστης ακόμη και ποιους χαρακτήρες έχει εισάγει ήδη. Έπειτα διαβάζουμε από το πληκτρολόγιο έναν χαρακτήρα και τον αποθηκεύουμε στην μεταβλητή char.

Κομμάτι 6

```
while [ ${#char} -gt 1 ]; do
                    echo "Wrong input"
                    echo -e "\n\nEnter a character"
                    read char
done
```

Στην συνέχεια ελέγχουμε αν η μεταβλητή char είναι όντως χαρακτήρας ελέγχοντας απλά το πλήθος των χαρακτήρων της μεταβλητής char. Αν δεν είναι χαρακτήρας, εμφανίζεται κατάλληλο μήνυμα λάθους και διαβάσματος καινούργιου χαρακτήρα. Ο βρόγχος εκτελείτε μέχρις ότου ο χρήστης εισάγει χαρακτήρα. Αν η μεταβλητή char είναι χαρακτήρας συνεχίζεται ομαλά η εκτέλεση του προγράμματος.

Κομμάτι 7

```
i=0
found=0
while [ $i -lt $WORD_LENGTH ]; do
	if [ $char = ${CHAR_PIN[$i]} ]; then
		STR_GUESSES[$i]=$char
		found=1

	fi
	let "i++"
done
```

Εδώ ελέγχουμε αν ο χαρακτήρας εισαγωγής περιέχεται πουθενά στην μυστική μας λέξη. Αν ναι τότε αντικαθιστούμε (στην αντίστοιχη θέση του χαρακτήρα της λέξης) τον χαρακτήρα με παύλα στον πίνακα πού εμφανίζει στην οθόνη την μυστική λέξη. Έπειτα αλλάζουμε την τιμή της μεταβλητής found η οποία παίζει το ρόλο της σημαίας.

Κομμάτι 8

```
if [ $found -eq 1 ]; then
let "GUESSES++"
fi
GUESSED=${GUESSED}$char
```

Σε αυτό το block εντολών ελέγχουμε μέσω της σημαίας found αν ο χαρακτήρας εισαγωγής υπήρχε στην μυστική λέξη. Αν ναι τότε αυξάνουμε τον μετρητή που μετράει πόσες προσπάθειες έχουμε. Αυτό έχεις ως αποτέλεσμα όταν μαντέψουμε σωστά έναν χαρακτήρα να παγώνει την αντίστροφη μέτρηση των διαθέσιμων λανθασμένων εισαγωγών χαρακτήρα. Τέλος προσθέτουμε τον χαρακτήρα εισαγωγής στην μεταβλητή GUESSED η οποία εμφανίζει στην οθόνη του χαρακτήρες που έχουμε εισάγει μέχρι τώρα.

Κομμάτι 9

```
TEMP=$(echo ${STR_GUESSES[@]} | sed 's/ //g')
if [ $TEMP = $WORD ]; then
	echo "The word is: $WORD"
	echo "YOU WON"
	break
fi
```

Εδώ αποθηκεύουμε προσωρινά τον “πίνακα με τις παύλες” που εμφανίζει στην οθόνη μας την μυστική λέξη στην μεταβλητή TEMP. Έπειτα ελέγχουμε αν η μεταβλητή TEMP είναι ίση με την μυστική λέξη. Αν είναι ίσες σημαίνει ότι ο χρήστης έχει μαντέψει σωστά πριν το πέρας των δέκα προσπαθειών την μυστική λέξη και έπειτα από την εμφάνιση των κατάλληλων μηνυμάτων, το πρόγραμμα τερματίζεται.

Κομμάτι 10

```
if [ $GUESSES -eq 0 ] ; then
        echo "Game over!! "
        echo "The word was: $WORD"
fi
```

Τέλος, έξω από την δομή επανάληψης, ελέγχουμε αν όντως η επανάληψη εκτελέστηκε δέκα φορές. Αυτό σημαίνει ότι ο χρήστης δεν κατάφερε να μαντέψει σωστά την μυστική λέξη και έτσι μετά την εμφάνιση κατάλληλων μηνυμάτων αποτυχίας, το πρόγραμμα τερματίζεται από μόνο του.

Ολοκληρωμένος και σχολιασμένος κώδικας

```
#!/bin/bash
#Το παιχνίδι κρεμάλα σε περιβάλλον κελύφους

WORDS=./words
IMAGES=./images

#Ελέγχουμε αν το αρχείο λέξεων υπάρχει και είναι προσπελάσιμο
if [ ! -r ${WORDS} ]; then
   echo "${WORDS} does not exist or is not readable" && exit 1
fi

#Μετράμε τις λέξεις που βρίσκονται στο αρχείο words
WORDSLN=$( wc -l ${WORDS} | awk '{ print $1}' )
#Δημιουργούμε έναν τυχαίο αριθμό, πάντα μικρότερο από τον αριθμό των λέξεων μας
```

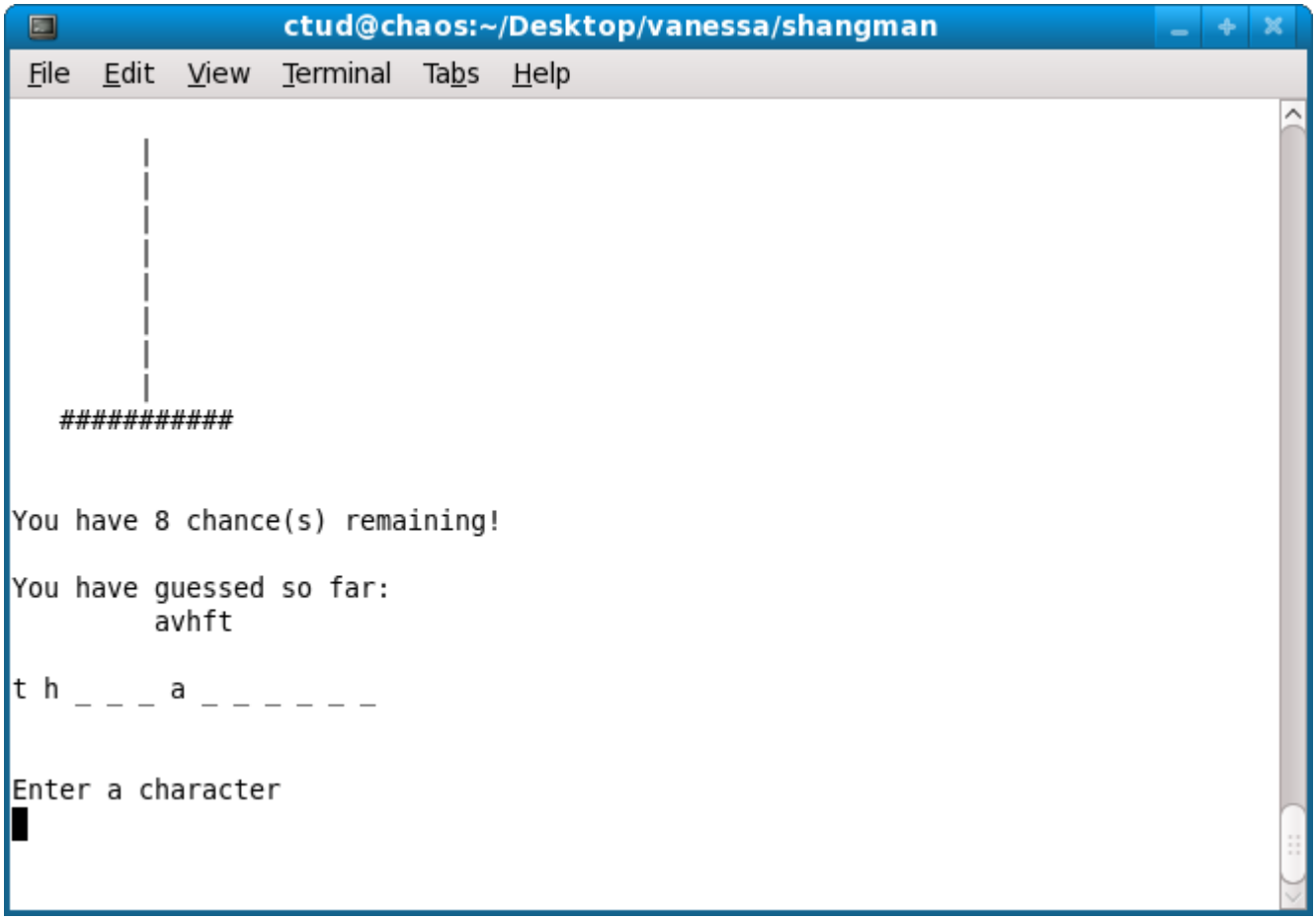
```
RANDOMN=$( awk -v l=$WORDSLN 'BEGIN { srand();
                        r = rand() * l;
                        printf( "%d", ( r + 1 ) ) }' )

#Επιλέγουμε την λέξη, με βάση τον τυχαίο αριθμό
WORD=$( awk -v l=$RANDOMN 'BEGIN { RS="";
                        FS="\n" } { print $l }' $WORDS)

#Βρίσκουμε το μέγεθος της λέξης
WORD_LENGTH=${#WORD}

#Αποθηκεύουμε τους χαρακτήρες έναν προς ένα στον πίνακα CHAR_PIN
for i in $( seq 0 $(( $WORD_LENGTH )) )
do
   CHAR_PIN[$i]=$(echo "${WORD}" | cut -c$(($i+1)) )
done



i=0
#Γεμίζουμε τον πίνακα STR_GUESSES με κάτω παύλες (_)
while [ $i -lt $WORD_LENGTH ]; do
        STR_GUESSES[$i]="_"
        let "i++"
done

GUESSED=""
flag=0
GUESSES=10

#Επανέλαβε όσες και οι διαθέσιμες προσπάθειες
while [ $GUESSES -gt 0 ] ; do

#Εμφάνηση εικόνας και πληροφοριών της υπάρχουσας κατάστασης
```

```
  clear
  cat ${IMAGES}/${GUESSES}
  echo -e "\n\nYou have ${GUESSES} chance(s) remaining!"
        echo -e "\nYou have guessed so far:\n\t $GUESSED"
echo -e "\n${STR_GUESSES[@]}"
echo -e "\n\nEnter a character"
read char

#Ελεγχος για το κατα πόσο η μεταβλητή char είναι χαρακτήρας
while [ ${#char} -gt 1 ]; do
                echo "Wrong input"
                echo -e "\n\nEnter a character"
                read char
done
i=0
found=0

#Ελεγχος για το αν ο χαρακτήρας περιέχεται στην μυστική λέξη
while [ $i -lt $WORD_LENGTH ]; do
        if [ $char = ${CHAR_PIN[$i]} ]; then
                STR_GUESSES[$i]=$char
                found=1

        fi
        let "i++"
done

#Αν ο χαρακτήρας περιέχεται στην μυστική λέξη επανέφερε τον μετρητή
if [ $found -eq 1 ]; then
let "GUESSES++"
fi

#Πρόσθεσε τον χαρακτήρα στου χαρακτήρες που έχουν εισαχθεί μέχρι τώρα
```

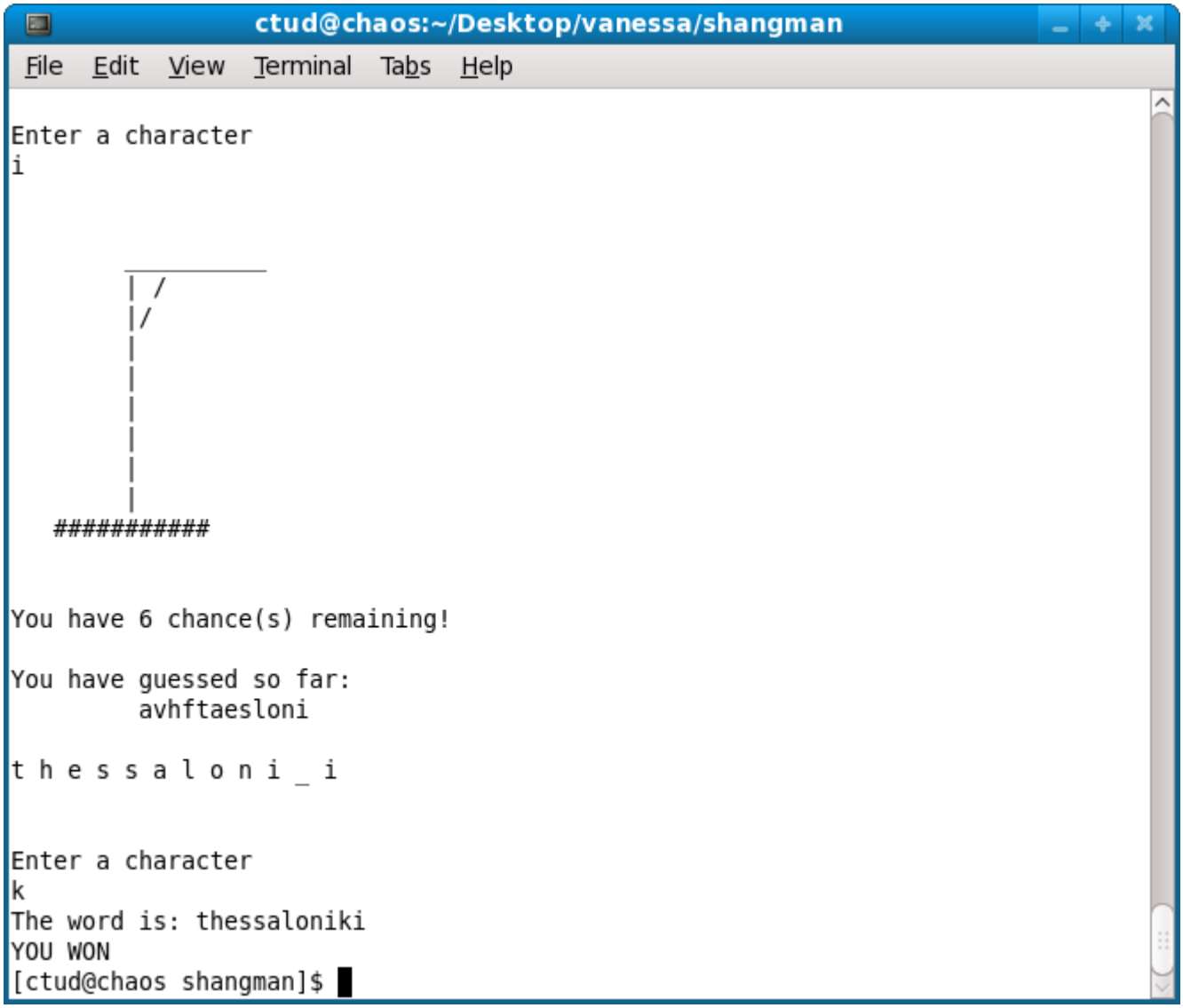
```
GUESSED=${GUESSED}$char

#Ελεγχος για το αν ο χρήστης έχει μαντέψει σωστά την μυστική λέξη
TEMP=$(echo ${STR_GUESSES[@]} | sed 's/ //g')
if [ $TEMP = $WORD ]; then
        echo "The word is: $WORD"
        echo "YOU WON"
        break
fi

let "GUESSES--"

done

#Ελεγχος για το αν έγιναν 10 επαναλήψεις, άρα ο χρήστης έχασε.
if [ $GUESSES -eq 0 ] ; then
        echo "Game over!!"
        echo "The word was: $WORD"
fi
```

Εκτέλεση προγράμματος

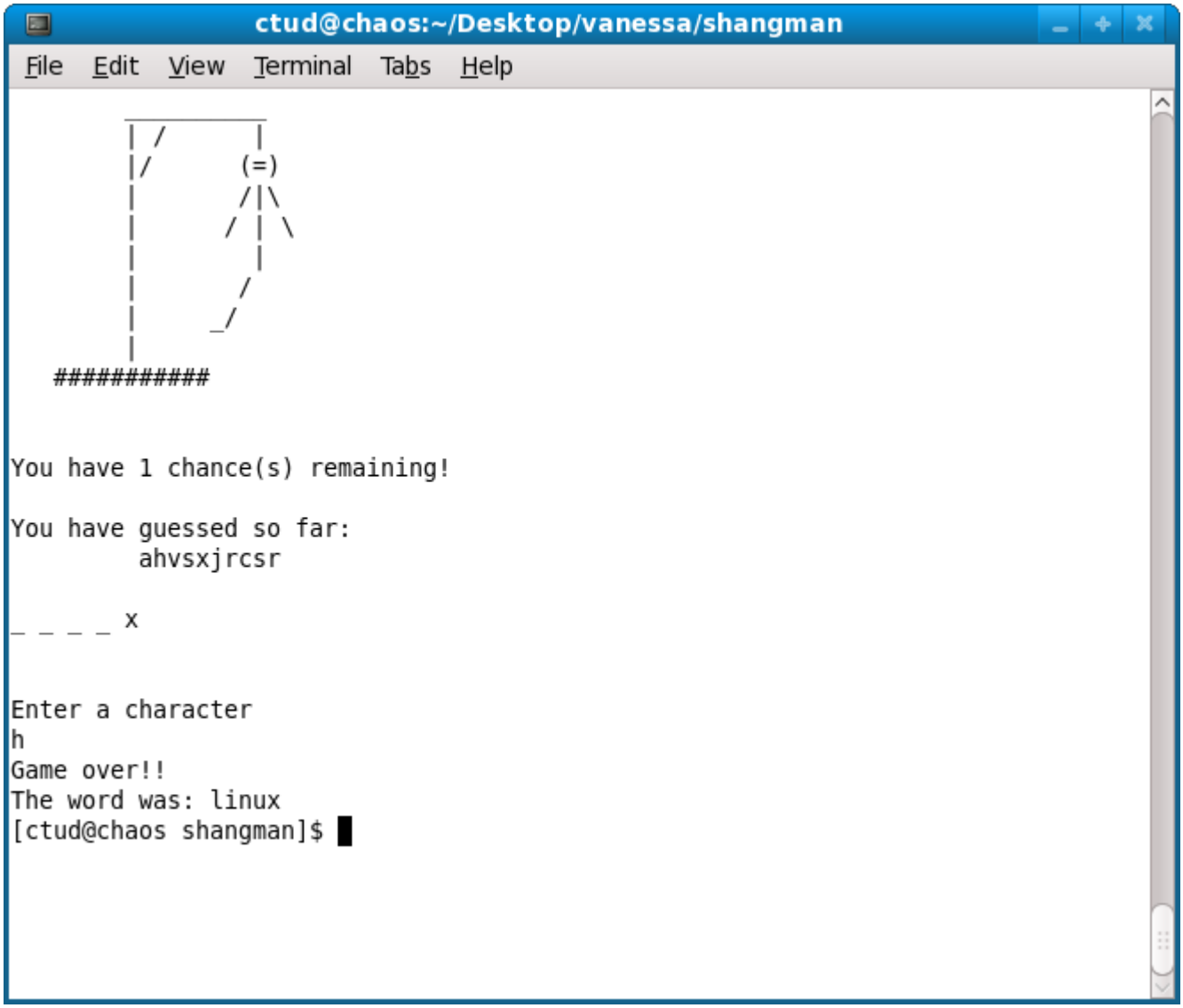
Παρακάτω παραθέτω μερικές εικόνες με μια σύντομη περιγραφή από την εκτέλεση του προγράμματος.



Σε αυτή την εικόνα απομένουν στον χρήστη ακόμη 8 προσπάθειες, έχει βρεί σωστά τους χαρατήρες t,h,a και έχει εισάγει μέχρι τώρα τους χαρακτήρες a,v,h,f και t.



Εδώ ο χρήστης έχει μαντέψει σωστά τα γράμματα της λέξης με μόλις τέσσερις λανθασμένες υποθέσεις.



Σε αυτή την περίπτωση ο χρήστης δεν κατάφερε να μαντέψει σωστά την μυστική λέξη και έτσι έχασε.

Βιβλιογραφία

- Wikipedia
- Σημειώσεις μαθήματος Λειτουργικών Συστημάτων 2