Relazione relativa allo sviluppo dell'applicazione Tubby Hamster.

Componenti del gruppo: Anna Chiara Aprile, Greta Sasso.

Analisi del problema

Si è deciso di realizzare un gioco per computer. L'idea è quella di realizzare un gioco il cui protagonista, un criceto con qualche chilo di troppo, dovrà correre per mantenere la sua linea e non dovrà cedere alla tentazione di cioccolato, ciambelle e gelati.

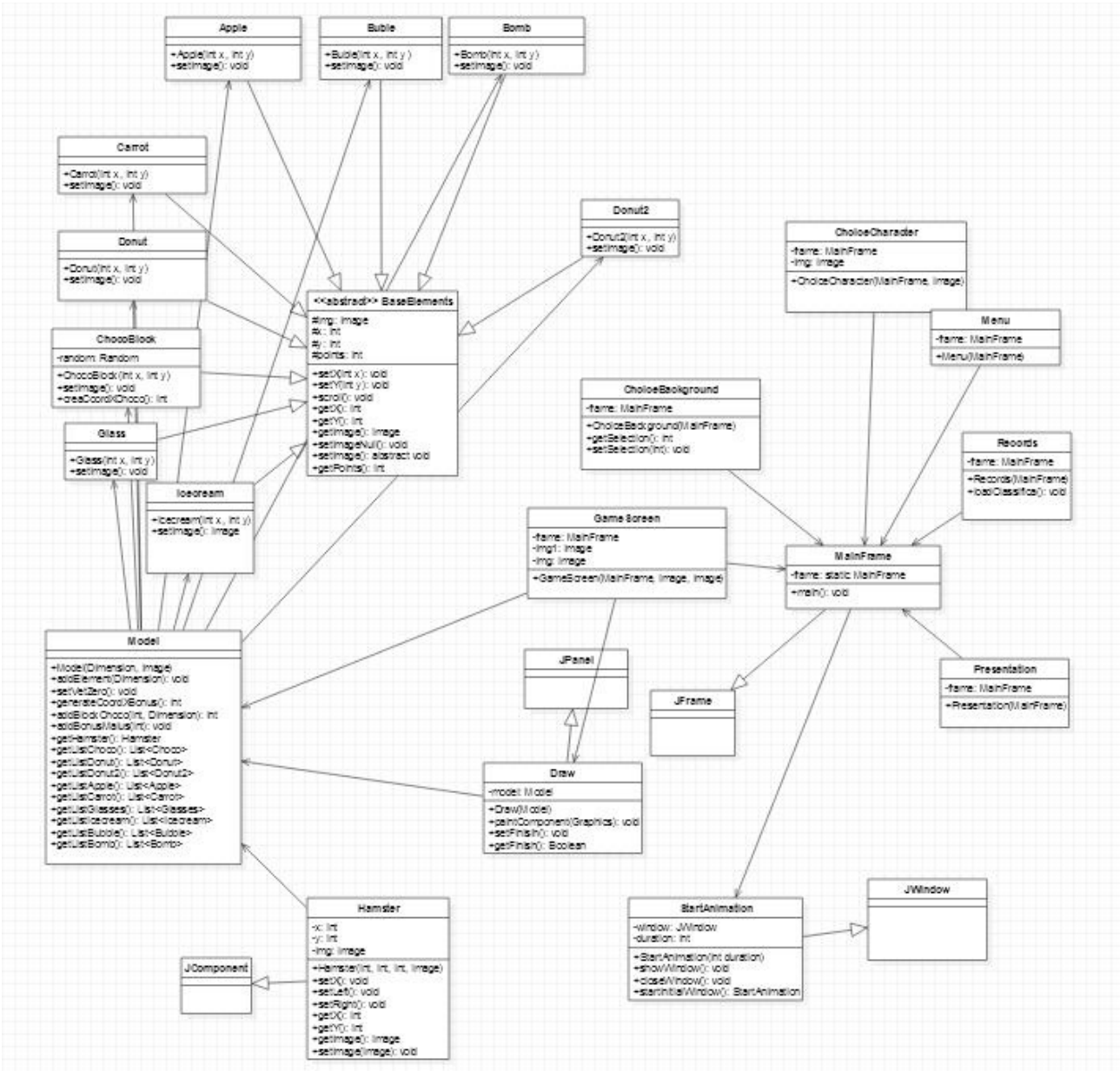
Il criceto dovrà quindi cercare di totalizzare più punti possibili, evitando di andare a collidere con dei blocchi di cioccolato che delimitano il percorso. Nello svolgimento della partita il criceto troverà degli oggetti che possono aiutare (bonus) o intralciare (malus) la sua missione.
Oggetti come mele, carote, bolle rappresentano i bonus, mentre ciambelle, gelati, lenti e bombe rappresentano i malus, ognuno con una propria caratteristica.
Le caratteristiche di questi oggetti possono variare dalla semplice alterazione (in positivo o in negativo) del punteggio, al cambio dello sfondo di gioco o persino alla modifica della velocità degli oggetti che scorrono.

Prima dell'avvio della partita , verrà data la possibilità all'utente di scegliere lo sfondo del gioco e il criceto con cui vuol affrontare la sfida, in modo da avere una esperienza di gioco personalizzata. È stata inoltre implementata la variazione della velocità del gioco e dei tempi degli effetti dei bonus/malus.

Ogni punteggio raggiunto, anche se negativo, sarà poi salvato in una classifica che si aggiornerà con i migliori risultati a ogni partita (ed avvio del programma), garantendo così la persistenza dei dati.

L'applicazione presenta quindi un menù iniziale, in cui l'utente potrà scegliere se avviare la partita, guardare la classifica dei migliori punteggi oppure uscire. Si è aggiunta inoltre una breve descrizione iniziale per aiutare l'utente a capire le meccaniche del gioco.

Inseriamo adesso il diagramma UML da cui siamo partiti:



Progettazione architetturale

Partendo dall'analisi dei requisiti del sistema, definiamo con la progettazione architetturale come impostare la struttura e quali saranno le relazioni principali.

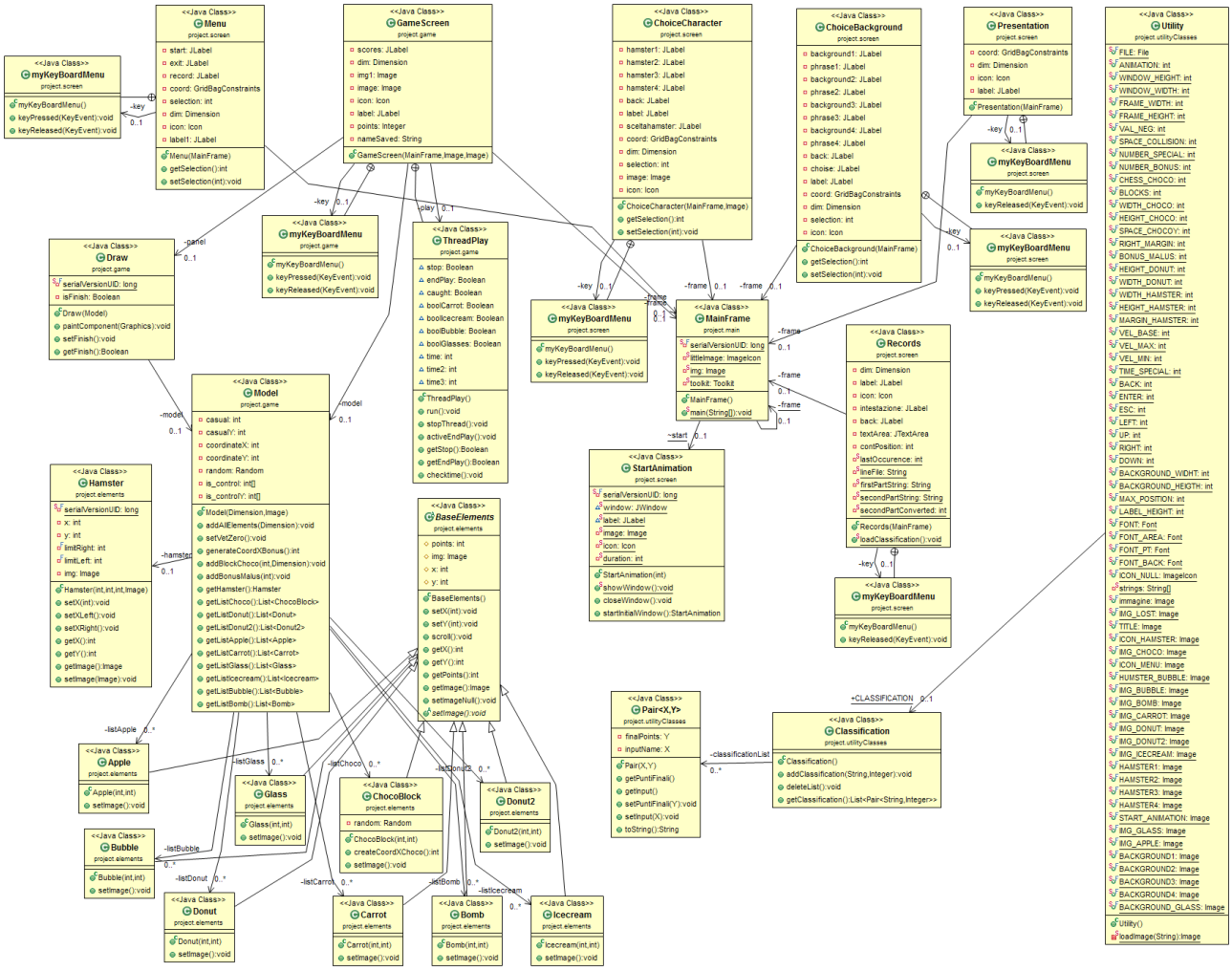
Abbiamo deciso di unificare la parte dei bonus/malus e degli altri elementi attraverso una classe astratta, poiché ogni oggetto ha molteplici caratteristiche e comportamenti simili.

Il programma parte da una schermata iniziale di introduzione, da cui successivamente l'utente passerà ad una schermata di menù dove avrà a disposizione un elenco di opzioni, fra le quali una contenente il gioco, la schermata dei record, oppure si potrà uscire dal programma.
L'apertura di ogni schermata istanzierà quindi una nuova classe che rappresenta la nuova interfaccia con il giocatore in cui verrà creata una label e posizionata nel frame. Per poter effettuare questo passaggio, ogni classe avrà bisogno di un riferimento alla classe che rappresenta il frame.

Abbiamo deciso di utilizzare il pattern di progettazione model-view-control per quanto riguarda parte dedicata al gioco al fine di separare la parte della visualizzazione dalla parte dei dati e da quella del controllo.
La schermata che lo contenente è la più importante poiché rappresenta il control del pattern. Ad essa si può accedere una volta effettuata la scelta dello sfondo e del personaggio. Come tutte le altre classi istanziate, crea una label che si andrà a sostituire alla precedente. Inoltre ha il compito di avviare il thread del gioco.
I dati degli elementi del gioco sono stati raggruppati nella classe BaseElement e specializzati nelle sue estensioni. I metodi per accedere ai dati e per elaborarli sono contenuti della classe Model , che incarna la prima tipologia del pattern usato.
Nella view del pattern, chiamata nel nostro caso Draw, inseriremo i metodi per visualizzare e disegnare i dati creati nella classe Model con le relative posizioni.
Il gioco è controllato infine dalla classe GameScreen che interpreta il ruolo del controller. al suo interno abbiamo un thread che si occupa dello scorrimento di ogni oggetto e del controllo delle collisioni e delle eventuali conseguenze .

Abbiamo deciso inoltre di rendere il mouse inaccessibile all'utente se non per quanto riguarda l'utilizzo delle finestre di dialogo. È quindi possibile passare da una schermata ad un'altra con il solo impiego della tastiera.

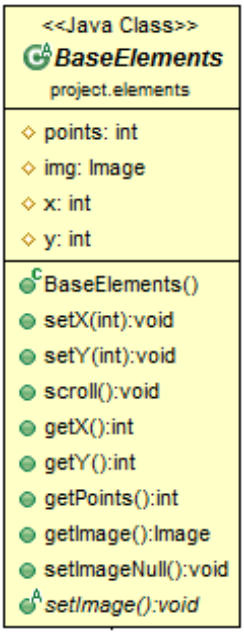
Presentiamo ora il modello dell'architettura del sistema tramite diagrammi UML:



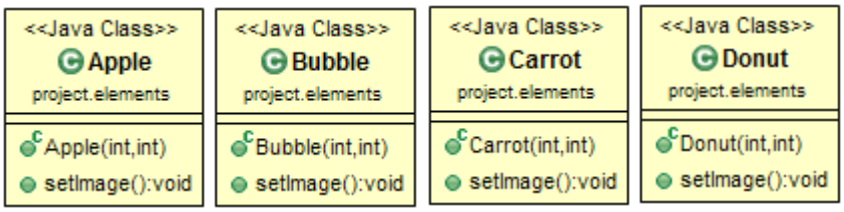
Descrizione degli aspetti principali:

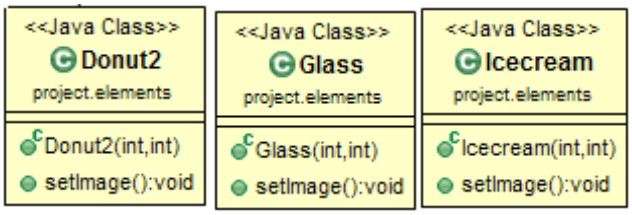
Classi che definiscono i dati degli elementi di gioco:

- **BaseElements**: Classe astratta, cioè una interfaccia che definisce in parte i suoi metodi. Essa modella i dati di tutti gli elementi del gioco (criceto, scacchi di cioccolato, mele, ecc) accomunando le proprietà comuni come l'immagine e le coordinate e i comportamenti fra i quali troviamo il metodo scroll che incrementa la coordinata y consentendo lo spostamento dell'oggetto in questione verso il basso.

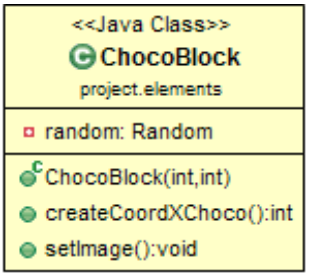


- **Apple**, **Bomb**, **Carrot**, **Donut**, **Donut2**, **Glass**, **Icecream**: sono tutte estensioni della classe BaseElements . Essa specializza l'oggetto descritto dal nome inizializzando i valori delle coordinate e della relativa immagine.

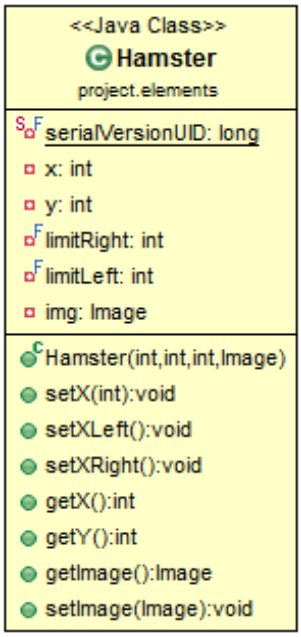




- **ChocoBlock**: è anch'essa una estensione della classe BaseElements. Essa rappresenta lo scacco di cioccolato per il quale la collisione da parte del criceto comporterà la fine della partita.

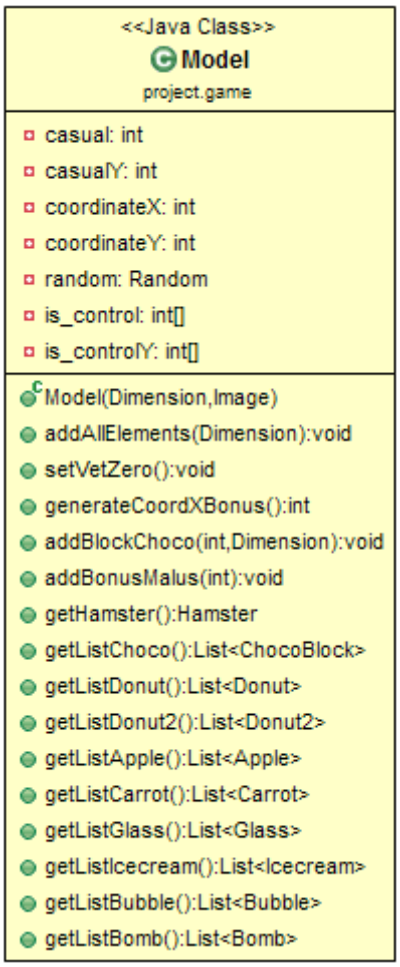


- **Hamster**: è l'ultima delle estensioni della classe BaseElements. Essa modella l'oggetto criceto, cioè il protagonista del gioco, definendo le variabili che rappresentano i suoi limiti e inizializzando le altre.

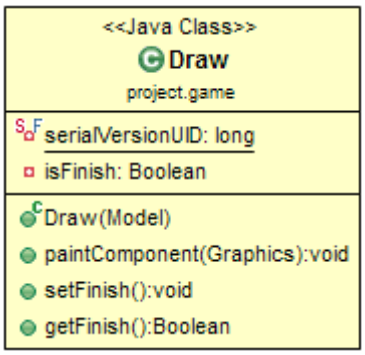


Classi che realizzano il gioco e costituiscono il pattern model-view-control :

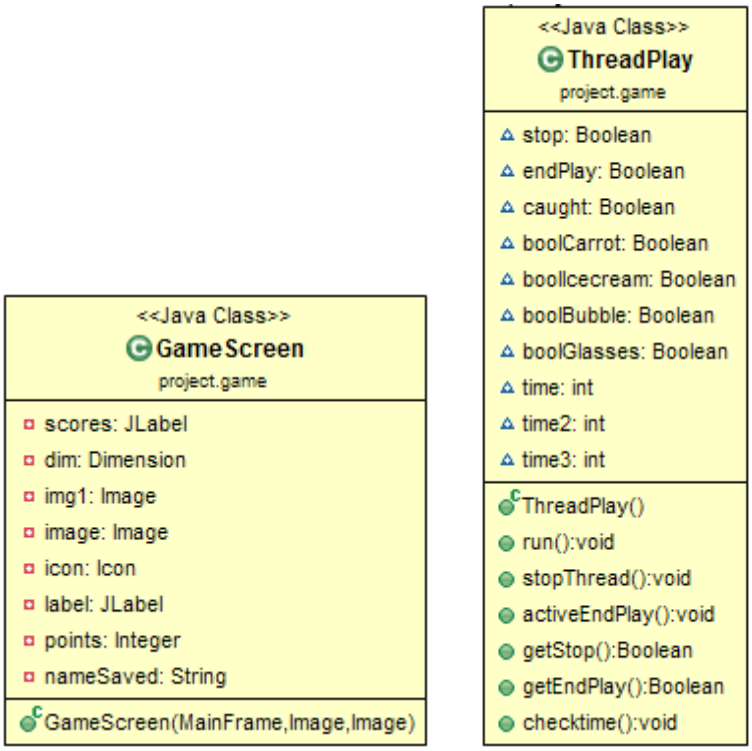
- **Model**: è una classe che istanzia tutte le liste contenenti i vari oggetti e crea le relative coordinate, definendo inoltre i metodi relativi.



- **Draw:** rappresenta la view del gioco. È la classe in cui vengono disegnati gli elementi del gioco. Questa classe estende un pannello a cui sono aggiunti i vari oggetti al fine di essere poi posizionati nella schermata che conterrà il gioco.

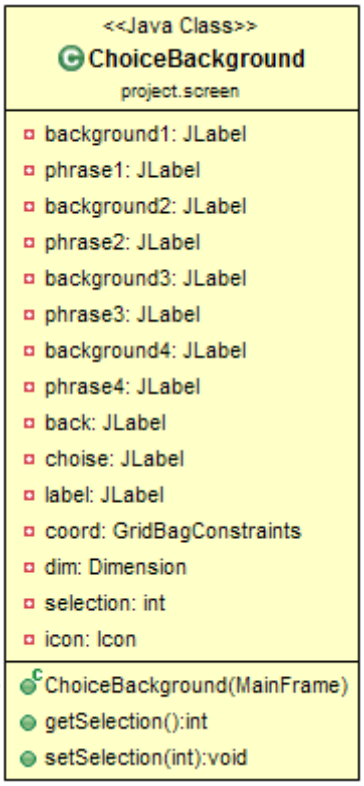


- **GameScreen:** Questa classe si occupa del Controllo del gioco, infatti essa istanzia e avvia un oggetto della classe ThreadPlay . In quest'ultima sono fatte variare le coordinate x e y di ogni oggetto e in contemporanea a questo avviene il controllo delle collisioni e vengono implementati gli effetti della cattura di ogni elemento (mele, carote, ecc) .



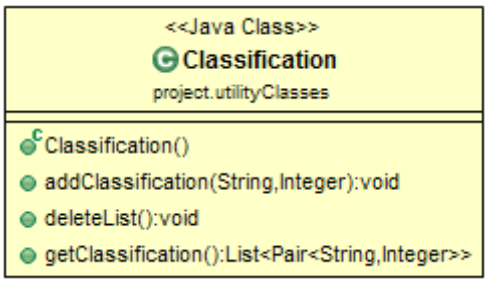
Classi che costituiscono le schermate del programma

- **ChoiceBackground**: Questa classe crea un pannello in cui viene implementata la scelta dello sfondo attraverso l'uso di un GridBagLayout . Il pannello , una volta creato, viene sostituito con quello della schermata precedente e posizionato direttamente sul frame.

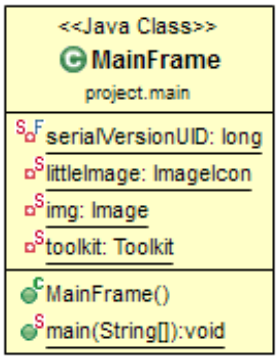


- **ChoiceCharacter**: Similare alla classe ChioceBackground , questa classe permette all'utente di scegliere il personaggio e quindi di navigare con l'uso della tastiera fra le varie opzioni e di confermare la sua scelta.

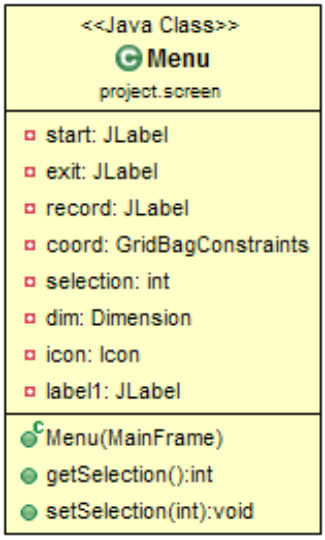
- **Classification**: Questa classe crea una lista di Pair in cui viene salvato il nome del giocatore con il rispettivo punteggio. Contiene inoltre il metodo utilizzato per confrontare gli elementi della lista, in modo da poter stilare una classifica ordinata in maniera decrescente.



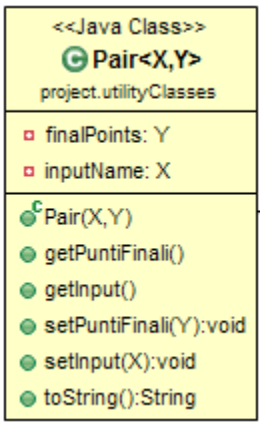
- **MainFrame**: Questa classe contiene il punto di partenza del programma e cioè il metodo main. Essa inoltre estende la classe JFrame e istanzia un oggetto di sè in modo da poter creare la base del programma.



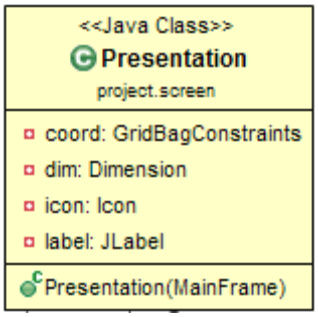
- **Menu**: Rappresenta la schermata contenente le opzioni per muoversi all'interno del programma . Essa viene creata tramite un pannello che si va a sostituire, come nelle precedenti situazioni, al frame.



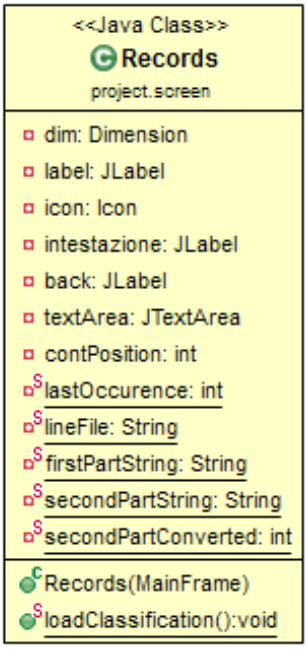
- **Pair**: Questa classe è stata creata per poter relazionare un giocatore al proprio punteggio. La struttura del Pair permette inoltre ad un giocatore di salvare più punteggi con lo stesso nome.



- **Presentation**: Questa classe, come deciso nella fase di analisi, si occupa semplicemente di introdurre il gioco con una breve spiegazione con le sue regole.

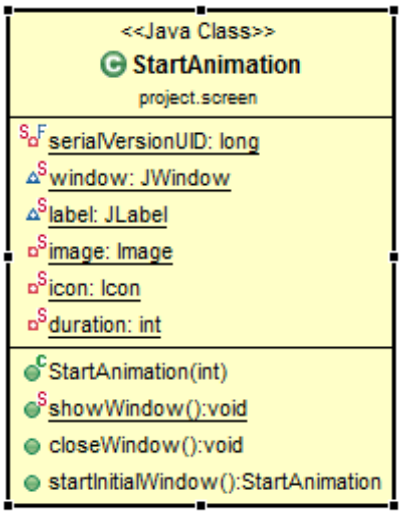


- **Records**: A questa classe, come alla classe GameScreen , si accede selezionando la medesima voce nel menu. Essa si occupa di creare un pannello contenente i risultati dei migliori punteggi .

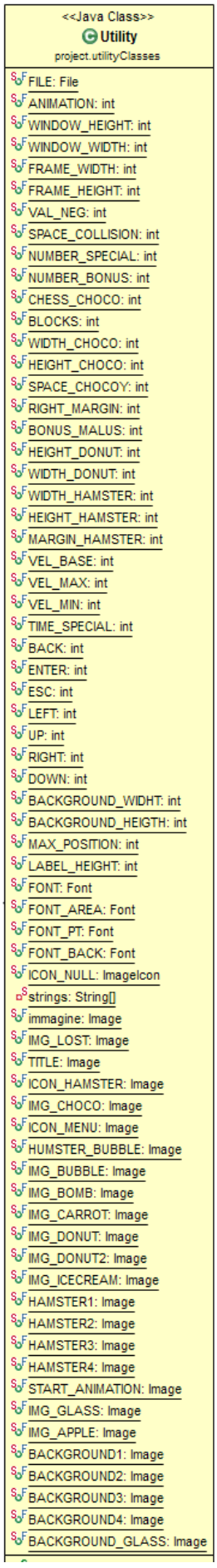


- **StartAnimation:** Questa classe rappresenta la prima schermata visualizzata in assoluto in cui verrà proiettata un'animazione introduttiva. Volendo visualizzare solo l'animazione , si è deciso che debba estendere la classe JWindow che attraverso

un thread controlla il tempo della sua comparsa e si sostituisce direttamente con schermata la cui classe è chiamata Presentation.



- **Utility**: Classe creata allo scopo di contenete tutte le variabili statiche che non vengono modificate all'interno del programma, come ad esempio le dimensioni degli oggetti, del frame, e altri magic number.


<<Java Class>>
Utility
project.utilityClasses
FILE: File
ANIMATION: int
WINDOW_HEIGHT: int
WINDOW_WIDTH: int
FRAME_WIDTH: int
FRAME_HEIGHT: int
VAL_NEG: int
SPACE_COLLISION: int
NUMBER_SPECIAL: int
NUMBER_BONUS: int
CHESS_CHOCO: int
BLOCKS: int
WIDTH_CHOCO: int
HEIGHT_CHOCO: int
SPACE_CHOCOY: int
RIGHT_MARGIN: int
BONUS_MALUS: int
HEIGHT_DONUT: int
WIDTH_DONUT: int
WIDTH_HAMSTER: int
HEIGHT_HAMSTER: int
MARGIN_HAMSTER: int
VEL_BASE: int
VEL_MAX: int
VEL_MIN: int
TIME_SPECIAL: int
BACK: int
ENTER: int
ESC: int
LEFT: int
UP: int
RIGHT: int
DOWN: int
BACKGROUND_WIDHT: int
BACKGROUND_HEIGTH: int
MAX_POSITION: int
LABEL_HEIGHT: int
FONT: Font
FONT_AREA: Font
FONT_PT: Font
FONT_BACK: Font
ICON_NULL: ImageIcon
strings: String[]
immagine: Image
IMG_LOST: Image
TITLE: Image
ICON_HAMSTER: Image
IMG_CHOCO: Image
ICON_MENU: Image
HUMSTER_BUBBLE: Image
IMG_BUBBLE: Image
IMG_BOMB: Image
IMG_CARROT: Image
IMG_DONUT: Image
IMG_DONUT2: Image
IMG_ICECREAM: Image
HAMSTER1: Image
HAMSTER2: Image
HAMSTER3: Image
HAMSTER4: Image
START_ANIMATION: Image
IMG_GLASS: Image
IMG_APPLE: Image
BACKGROUND1: Image
BACKGROUND2: Image
BACKGROUND3: Image
BACKGROUND4: Image
BACKGROUND_GLASS: Image

Organizzazione in package

Procediamo adesso con la stesura dell'analisi della suddivisione in package:

- **project.screens**: contiene l'insieme delle classi che si occupano dell'implementazione di tutte le schermate che fungono da interfaccia con l'utente:
  - **ChoiceBackground**
  - **ChoiceCharacter**
  - **Classification**
  - **Menu**
  - **Presentation**
  - **Records**
  - **StartAnimation**

- **project.game**: questo package contiene le classi che costituiscono il gioco e cioè le tre tipologie definite dal pattern model view control:
  - **GameScreen**
  - **Draw**
  - **Model**

- **project.elements**: Abbiamo raggruppato in questo package tutte le classi che contengono e descrivono tutti gli elementi, incluso la classe astratta che definisce le variabili e I metodi e che viene estesa dalle altre classi internet al package:
  - **BaseElements**
  - **Apple**
  - **Bomb**
  - **Bubble**
  - **Carrot**
  - **Chocoblock**
  - **Donut**
  - **Donut2**
  - **Hamster**
  - **Glass**
  - **Icecream**

- **project.images:** questa classe è stata creata per contenere tutte le immagini usate nel programma.

- **project.main**: è costituito dalla classe MainFrame contenente il metodo main necessario per l'avvio del programma:
  - **MainFrame**

- **project.utilityclasses**: Abbiamo inoltre raggruppato le classi non inerenti ai gruppi precedenti ma comunque di utilità:
  - **Classification**
  - **Pair**
  - **Utility**

Suddivisione del lavoro

Il progetto è stato suddiviso in questo modo tra i componenti del gruppo:

- **Greta sasso :** si è occupata della modellazione dei dati attraverso la classe model, della gestione del thread e della rilevazione delle collisioni del criceto con i blocchi di cioccolato e la cattura degli elementi. Si è occupata inoltre della realizzazione delle immagini e di alcune classi che estendono BaseElements. Ha provveduto all'eliminazione del mouse all'interno della schermata del programma e ha implementato la selezione nei menu delle classi ChiocеBackground , ChoiceCharacter e Menu.

- **Anna Chiara Aprile:** si è occupata della gestione degli effetti provocati dalla cattura degli elementi del gioco all'interno del Thread principale, come ad esempio la modifica dello sfondo durante la partita. Si è inoltre occupata delle interfacce grafiche delle schermate. Ha implementato il salvataggio dei punteggi delle partite e la stesura della classifica nella classe Records. Infine ha provveduto all'inserimento delle finestre di dialogo per l'uscita dal gioco tramite la tastiera o icona di uscita.

Abbiamo lavorato insieme su alcune parti del progetto che riguardano in particolare l'implementazione della classe astratta BaseElements, GameScreen, delle classi che implementano il KeyListener e del MainFrame. Verranno poi specificate nel dettaglio le eventuali suddivisioni del lavoro all'interno di queste classi.

Progettazione di dettaglio: parte di Greta Sasso

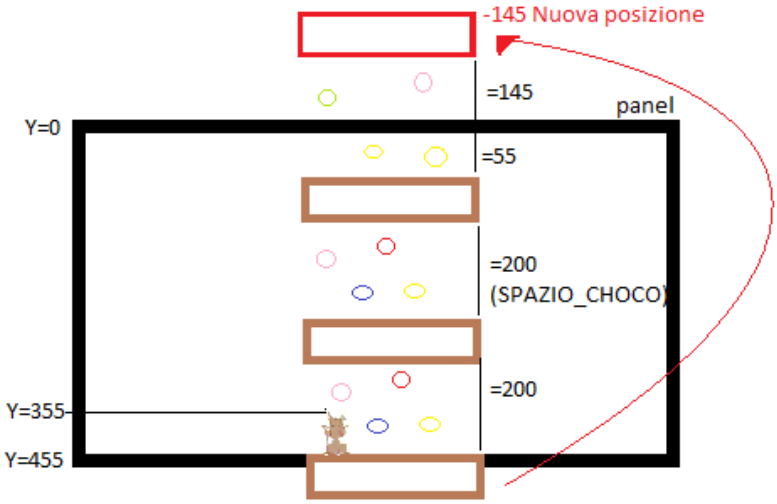
Partiamo dall'implementane delle classi:

- **Apple, Donut, Donut2 :** Queste classi estendono e quindi ereditano tutti i metodi e le variabili protected e public della classe BaseElements. Nel loro costruttore sono state inizializzate le coordinate degli oggetti e le loro immagini. È stato inoltre definito il metodo astratto della superclasse setImage. Quest'ultimo ha il compito di impostare il valore dell'immagine corretta, poiché successivamente, nel thread del gioco, con la cattura dell'elemento in questione, sarà settata l'immagine con un valore nullo per occultarla momentaneamente.

- **Chocoblock:** Questa classe, come la precedente, estende BaseElements inizializzando nel costruttore le sue coordinate e l'immagine.
  Per quanto riguarda la coordinata X è stato creato un metodo createCoordXChoco, chiamato poi nel costruttore. Esso calcola in modo casuale la coordinata x partendo da un numero che può variare da 1 a Utility.CHESS_CHOCO (che corrisponde al numero di blocchi di cioccolato massimi inseribili lungo l'asse orizzontale) moltiplicato per il numero di pixel della larghezza dell'immagine, ottenendo in questo modo il calcolo della coordinata precisa per al massimo 15 possibili blocchi di cioccolato.

- **Hamster:** Ulteriore Classe che eredita da BaseElements metodi e campi. Essa si occupa di inizializzare i dati relativi al criceto per quanto riguarda l'immagine, le coordinate e i limiti. A differenza delle altre classi sopra descritte, presenta due variabili nuove, limitRight e limitLeft che corrispondono ai valori minimi e massimi che la coordinata X del criceto può assumere quando viene modificata mediante l'uso della tastiera.

- **GameScreen:** Questa è la classe principale del programma in cui viene definito il control del gioco. Alla classe vengono passati come parametri i valori delle immagini dello sfondo e del criceto scelti in precedenza e assegnati agli appositi campi. Viene creata poi una label, con layout di tipo BorderLayout e come icona l'immagine di sfondo, che viene posizionata al posto della precedente schermata nel frame. Ad essa viene aggiunto nella parte centrale il pannello creato istanziando la classe Draw, e nella parte sud una label contenente il punteggio del gioco. Alla fine della classe viene Istanziato e avviato tramite il metodo start() un oggetto della classe MyThread . Quest'ultima implementa il vero controller del gioco, cioè il thread.
  Il thread rimane attivo finchè la variabile booleana, chiamata stop , avrà valore false, in quanto il suo valore diventerà true una volta terminata la partita, per una collisione o per volontà dell'utente tramite pulsante di uscita. All'interno del ciclo viene implementato l'effettivo controllo del gioco. Ho deciso di analizzare per ogni ciclo del thread tutte le liste contenenti gli elementi del gioco e di confrontare le coordinate di ognuna di queste con quelle del criceto. All'interno di ognuno di questi cicli, per ciascun elemento , viene prima chiamato il metodo scrool() che è responsabile dello scorrimento verso il basso, incrementando la coordinata y. Immediatamente si verifica e la coordinata dell'elemento abbia o meno superato l'altezza della finestra. Nel caso positivo, si modifica la coordinata x richiamando i metodi per calcolarla in modo casuale contenuti nel Model, e la coordinata y riportandola in cima. In particolare:



  Si effettua poi il controllo sulle possibili collisioni, fatto in modo tale che il criceto possa sovrapporsi di qualche pixel ai bordi dell'immagine (Utility.SPAZIO_COLLISIONI) per avere un'esperienza di gioco meno rigida. Le collisioni sono studiate per i quattro lati del criceto. Sono quindi considerate: la coordinata x del criceto, che rappresenta il punto di partenza sull'asse delle x della sua immagine, la coordinata x sommata alla larghezza del personaggio, che corrisponde al valore dell'angolo di destra, e lo stesso per la coordinata y. Per quanto riguarda i bonus malus, essi vengono catturati se la parte centrale del criceto corrisponde con la loro. In particolare si è diviso sia il valore dell'altezza che della larghezza per 2/3 e per 1/3 per avere le coordinate delle parti interne degli elementi considerati.
  Infine nella classe myKeyBoardMenu ho implementato lo spostamento del criceto mediante l'utilizzo della tastiera. Con il tasto della freccia di sinistra verrà chiamato il metodo del criceto che diminuirà il valore della coordinata x, al contrario con la freccia di destra, la coordinata verrà aumentata.

- **Model:** Questa classe rappresenta il model del pattern model-view-control in quanto modella i dati di interesse. Considerando che nel gioco ci sarà una presenza elevata di

elementi dello stesso tipo , ho deciso di raggruppare gli oggetti in ArrayList. Ho inizializzato quindi una lista per ogni tipo di elemento che il criceto dovrà catturare e una variabile per il criceto di tipo Hamster. Ho deciso di disporre gli elementi in tre blocchi, ognuno dei quali composto da una riga di ostacoli di cioccolato e nello spazio rimanente dagli altri elementi. I margini composti dai blocchi di cioccolato, anch'essi inseriti in lista, hanno coordinate fisse nel tempo, al contrario dei blocchi centrali la cui coordinata x viene di volta in volta ridefinita. Stessa cosa è applicata agli elementi del gioco la cui variabile x viene modificata in modo casuale sfruttando due vettori appositi per controllare che nella nuova coordinata non ci sia già presente un altro oggetto. Ognuno dei blocchi è organizzato da un ciclo presente nel metodo addAllElements . All'interno del ciclo, per ogni blocco, saranno chiamati tre metodi addBlockChoco, setVetZero e addBonusMalus.
addChocoBlock si occupa, come da titolo, del calcolo delle coordinate dei blocchi di cioccolato e della loro aggiunta nella listChoco. Esso, in un ciclo che si ripete per ogni riga che si viene a creare, crea prima l'elemento al bordo di sinistra, poi a quello di destra e infine , tramite un ulteriore ciclo, crea i blocchi centrali usando di volta in volta la funzione createCoordXBonus().
Il metodo setVetZero invece ha il compito di settare ogni valore delle celle dei vettori, che controllano le presenze, a 0, per ogni blocco, in modo che in quello nuovo si abbia la situazione di non aver nessun elemento.
Infinte il metodo addBonusMalus, come quello descritto in precedenza, si occupa di aggiungere, per ogni tipo di oggetto, l'elemento nella corrispondente lista. Usa anch'esso il metodo generateCoordXBonus e , oltre a controllare le presenze sull'asse orizzontale, controlla che l'elemento non vada a soprapporsi in linea verticale.
Il metodo generateCoordXBonus si occupa di calcolare la coordinata nuova degli elementi in modo casuale e controlla che non sia la stessa di un oggetto già creato in quel punto. Nell'eventualità ripete il calcolo fino a ottenere un risultato che soddisfi le condizioni richieste. La coordinata x deve avere un valore compreso fra i bordi del cioccolato.

- **Eliminazione del mouse:** Per eliminare il mouse dalla schermata del programma ho semplicemente assegnato al cursore del mouse una immagine inesistente.

- **Menu di selezione:** Ho implementato la modalità di selezione delle opzioni di un menu nelle classi ChoiceBackground, ChoiceCharacte e Menu. Ho dichiarato una variabile selection che varia da 1 al numero di opzioni e l'ho inizializzata sulla prima voce, ponendola uguale a 1.
Nella classe myKeyBoardMenu , alla pressione di una delle frecce direzionali, viene modificato, all'interno di un costrutto switch, il valore della variabile selection il colore delle scritte per meglio risaltare l'opzione selezionata rispetto alle altre. In un nuovo costrutto switch, alla pressione del tasto invio a seconda del valore della variabile selection , verrà istanziata la classe corrispondente alla voce del menu.

Progettazione di dettaglio: parte di Anna Chiara Aprile

- **ChoiceBackground:** Grazie a questa classe viene data la possibilità al giocatore di scegliere lo sfondo che vuole applicare al gioco. Viene istanziata all'interno del frame una label con layout di tipo BorderLayout contenente i quattro sfondi tra cui poter scegliere. È stato utilizzato un GridBagConstraints per creare una griglia per la disposizione ordinata delle quattro immagini.

- **ChoiceCharacter:** Grazie a questa classe viene data la possibilità al giocatore di scegliere il personaggio con il quale vuole giocare. All'interno del frame, passato per riferimento, è stata istanziata una label contenente le quattro immagini dei criceti, tra cui poter scegliere. Anche in questo caso, come nella classe ChoiceBackground, è stato utilizzto un GridBagConstraints per la creazione di una griglia immaginaria al fine di ottenere una disposizione equilibrata degli elementi.

- **Classification:** Questa classe è stata creata per poter stilare una classifica dei migliori punteggi raggiunti. Nel costruttore di questa classe viene inizializzata una lista di Pair, che conterrà le coppie giocatore-punteggio che verranno create in una stessa sessione di gioco.
  Per aggiungere elementi alla lista viene creato ogni volta un nuovo Pair, tramite l'utilizzo del metodo addClassification(), che ha come parametri una stringa contenente il nome, che si è deciso di salvare, e il punteggio totalizzato.
  È stato inoltre implementato un metodo che restituisce una lista ordinata in modo decrescente. Per fare ciò, all'interno del metodo, viene creata una nuova lista a cui vengono passati tutti i valori della precedente e , tramite l'invocazione del Comparator, si procede all'ordinamento.

- **Menu:** Questa classe rappresenta il menu di gioco dell'applicazione. Presenta tre opzioni tra cui scegliere. La prima, ovvero l'opzione di start, fa partire il gioco, la seconda, Records, permette di visualizzare la classifica, e la terza, Exit, comporta l'uscita dal gioco e quindi la chiusura dell'applicazione. Anche in questa classe è stato usato un GridBagConstraints per incolonnare e centrare le tre label contenenti le tre opzioni di scelta.

- **Presentation:** Questa classe contiene una breve legenda del gioco. È stata creata una TextArea contenente una breve descrizione del gioco ed è stata creata una legenda dei bonus/malus tramite l'utilizzo di label disposte grazie all'aiuto del GridBagConstraints. Questi elementi infine sono stati aggiunti ad una label con layout di tipo GridBagLayout.

- **Records**: questa classe è accedibile tramite il menu, scegliendo l'opzione Records. In questa schermata viene mostrata la classifica.
  Quando l'utente vi accede, viene aperto il file contenete la classifica in lettura. Ogni riga del file viene riportata su una TextArea fino ad un massimo di 14 righe. Alla label di sfondo sono state inoltre aggiunte altre due label, una che mostra il titolo della schermata, e l'altra che visualizza l'opzione per tornare indietro al menu. In questa classe è stato anche implementato il metodo loadClassification() che viene richiamato nella classe GamesScreen. Quando viene invocato questo metodo viene aperto il file in lettura. Ogni stringa del file viene suddivisa in due parti, nome e punteggio, e viene aggiunta alla lista di Pair precedentemente creata, che in questo momento conterrà tutti i vecchi salvataggi. A questo punto nella lista viene aggiunto anche il nuovo Pair. Per stilare una nuova classifica completa e ordinata viene cancellato il file e creato uno nuovo. La lista viene ordinata e ogni elementi della lista, grazie anche all'ausilio dell'Iterator, viene scritto sul nuovo file e sarà caricato poi nella TextArea della classe Records. Infine viene chiuso il file. In questa classe si provvede inoltre alla cancellazione degli eventuali elementi presenti nella lista, cioè nel caso in cui il giocatore effettui più partite, senza uscire dall'applicazione.

- **StartAnimation:** Questa classe, che estende JWindow rappresenta la finestra di avvio del gioco. Per la sua realizzazione , è stata utilizzata una label a cui è stata adattata l'immagine di presentazione.

Tramite l'utilizzo del metodo setLocation() la finestra è stata allocata al centro del desktop, nello stesso posto dove poi successivamente partirà l'applicazione.
Grazie all'utilizzo di un thread è stato implementato il metodo che permette alla finestra di mostrarsi solo per un periodo di tempo ben definito. Una volta chiusa la finestra il gioco si avvierà.

- **Draw:** Questa classe rappresenta la view del pattern model-view-control. In questa classe vengono disegnate tutte le componenti grafiche del gioco. La view riceve gli aggiornamenti dal model e li mostra all'utente. È per questo infatti che nel costruttore della classe Draw il Model viene passato come parametro.

- **Bomb, Bubble, Carrot, Glass, Icecream:** Queste classi estendono la classe BaseElements. Nel loro costruttore sono state inizializzate le coordinate dell'oggetto e le rispettive immagini.
  É stato inoltre definito il metodo astratto della superclasse setImage che ha il compito di impostare il valore dell'immagine corretta.

- **Pair:** Questa classe è stata creata per poter relazionare ogni giocatore al punteggio totalizzato, anche se il nome è già stato utilizzato. Sono stati implementati metodi di set, get e toString.

Progettazione di dettaglio: parti comuni

Alcune parti del progetto sono state affrontate insieme. Nella progettazione di dettaglio singola , abbiamo specificato che in alcune classi, di cui si occupa una persona , l'altra ha apportato delle piccole modifiche (ad esempio nel Menu con la sola selezione implementata da Greta sasso e il resto da Anna Chiara Aprile) .

La classe MainFrame estende JFrame. Essa contiene il metodo Main necessario per avviare il programma e internamente ad esso crea un oggetto di tipo MainFrame in modo da avviare il frame. In questa classe sono impostate le dimensioni della finestra ed è stata disabilitata la possibilità di ridimensionarlo.

Nella classe GameScreen , cui ci siamo suddivise vari compiti, abbiamo lavorato insieme sull'implementazione del codice per la modifica della velocità del gioco e dei tempi degli effetti delle catture di ogni elemento. Per fare questo abbiamo creato una apposita variabile, chiamata mol, ottenuta dalla divisione dei punti per un certo numero, in modo tale che ogni volta in cui si totalizzano quel numero di punti, si incrementino le variabili velocità e tempo. In particolare i tempi variabili sono stati creati per gli oggetti carota e gelato , che aumentano e diminuiscono la velocità.

È stata creata inoltre una classe chiamata Utility che raccoglie tutte le variabili statiche che non vengono modificate nel corso del programma, come tutte le immagini, le dimensioni del frame, i codici corrispondenti alle frecce, ecc.. Ognuno di questi campi viene chiamato direttamente col nome della classe ad esempio Utility.IMG_HAMSTER.
Le immagini sono state caricate, sfruttando un vettore contenente tutti i nomi con l'estensione inclusa, in un vettore di Image, in modo da sfruttare gli indici di quest'ultimo per assegnare a ogni variabile la relativa immagine.

Testing

Per la fase di testing del gioco si sono effettuate numerose partite col fine di provare tutte le possibili situazioni. Ad esempio si è provato a muovere il criceto oltre il margine del frame, anche con gioco in pausa. Si sono inoltre testati tutti gli effetti speciali catturati uno dopo l'altro in diverso ordine. Abbiamo controllato inoltre il salvataggio, con valori negativi oppure uguali ad altri già presenti.
Abbiamo inoltre utilizzato il plug-in "checkstyle" per correggere eventuali errori di forma nel codice e nella javadoc.
Non abbiamo riscontrato problemi rilevanti se non una scarsa reattività della tastiera e un problema alla pressione del tasto invio nelle finestre di dialogo.

Note Finali

Il processo di sviluppo è stato lineare ed è rimasto coerente con quanto deciso sin dall'inizio. Siamo inoltre riuscite a soddisfare gli eventuali punti opzionali definiti al momento della proposta del progetto sul forum.

Il lavoro di gruppo è proceduto bene con una buona comunicazione e collaborazione. Siamo soddisfatte del lavoro che siamo riuscite a ottenere e sviluppare.