|  |  |
| --- | --- |
| CSE272: Programming ll |  |

# Circus of Plates - Game

May 21, 2017

|  |  |
| --- | --- |
| Overview | * It is single player-game in which each clown carries two stacks of plates, and   there are a set of colored plates queues that fall down and he tries to catch them,  if he manages to collect three consecutive plates of the same color, then they are vanished  and his score increases. You are free to put rules ending the game.  The purpose of this project was to implement all the design patterns(with one extra pattern). |
| Class Diagram |  |
| Singleton | * Ensure a class has only one instance, and provide a global point of access to it. * Encapsulated "just-in-time initialization" or "initialization on first use".   Public static View getinstance(){  If(view==null){  View = new View();  }  Return view;  }  It was used with the viewer and the world classes since its only needed once. |
| Iterator | * Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation. * The C++ and Java standard library abstraction that makes it possible to decouple collection classes and algorithms. * Promote to "full object status" the traversal of a collection. * Polymorphic traversal. * It was used with checking the plates above each other, since you dnt know the type of list that the plates are saved in. |
| Dynamic Linkage | * Defines procedure of arbitrary loading and usage of classes at runtime - with only requirement for the classes, that they implement an interface, known at compile-time. Dynamic linkage simply reduces compile-time restrictions on behalf of more run-time freedom regarding class implementation. The procedure allows change in implementation of classes at run-time, without neither re-compiling nor stopping the program. * Was used to load shape classes dynamically to the game. |
| Snapshot | * Without violating encapsulation, capture and externalize an object's internal state so that the object can be returned to this state later. * A magic cookie that encapsulates a "check point" capability. * Promote undo or rollback to full object status. * Snapshot was used to pause and resume the game. |
| State | * Allow an object to alter its behavior when its internal state changes. The object will appear to change its class. * An object-oriented state machine * wrapper + polymorphic wrappee + collaboration * state was used to switch between levels…..level1 and 2 and 3 |
| Factroy | * factory was used for continues production of shape object |
| Strategy | * Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it. * Capture the abstraction in an interface, bury implementation details in derived classes. * Strategy was used to switch between worlds(“levels”). |
| Flyweight | * Use sharing to support large numbers of fine-grained objects efficiently. * The Motif GUI strategy of replacing heavy-weight widgets with light-weight gadgets. * Flyweight was used when creating many shapes that share similar colours in order to decrease the time required to load shapes |
| Observer | * Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. * Encapsulate the core (or common or engine) components in a Subject abstraction, and the variable (or optional or user interface) components in an Observer hierarchy. * The "View" part of Model-View-Controller. * Observer object was made to watch the plates and the clown .when the clown moves dx the plates will move dx too. |
| Mode-view-controller | * The controller was used for the score viewer. the score viewer was made to view the game itself(which is the view)once the level changes the screviewer call viewer class(mode)to swtich between levels(state). |
| Sample Run |  |
| Sequence Diagarams |  |

# References