_object: parseDir

Content:
```
public:
    void parseLib(const char *, const std::string = "lib/");
    void parseGame(const std::string = "games/");
    void changeLib(const int);
    const char *getCurrent();

    std::vector<std::string> _list;
private:
    void recupContent(const std::string);
```

Responsabilities
Lit et récupère le contenu des dossiers lib et game.
Gère quelle librairie est en cours de lecture.

_object: dll<T>

Content:
```
public:
    dll(const char *, const char *);
    ~dll();

    T *so;
private:
    void *handler;
```

Responsabilities
Charge une librairie et accès par la variable "so".

_enum: arcade

```
namespace state {
    typedef enum state_e {
        Username,
        Menu,
        Game,
        End,
    } state_t;
}

namespace key {
    typedef enum key_e {
        noaction,
        f2,
        f3,
        f4,
        f5,
        enter,
        escape,
        backspace,
        z,
        q,
        s,
        d,
    } key_t;
}
```

_interface:IDisplayModule

Content:
```
class  IDisplayModule {
    public:
        virtual ~IDisplayModule() {};
        virtual void doEvent(std::string &) = 0;
        virtual void clean() = 0;
        virtual bool isOpen() const = 0;
        virtual void display() = 0;
        virtual void init(const arcade::state::state_t) = 0;
        virtual void draw(const std::vector<std::string> &) = 0;

        virtual arcade::key::key_t const &getKey() const = 0;
        virtual arcade::state::state_t const &getState() const = 0;
        virtual void setKey(const arcade::key::key_t) = 0;
        virtual void setState(const arcade::state::state_t &s) = 0;

  virtual void setName(std::string &) = 0;
};
```

Responsibilities
Interface d'accès aux librairies graphique.

_object:Core

Content:
```
public:
    Core(const char *);
    ~Core();
    dll<IDisplayModule> *lib;
    dll<IGame> *game;
    parseDir _lib;
    parseDir _game;

    void setState(const arcade::state::state_t &e);
    const arcade::state::state_t &getState() const;

    void launchGame();
private:
    void gameInput();
    void changeLib(const int);
    void changeGame(const int);
    void loadGame();
    void swapLib();
    void doGame();
    void loop();
    void setupKeyGame();
private:
    std::string username;
    arcade::state::state_t _state;
    std::map<arcade::key::key_t, int (IGame::*)(void)> _inputGame;
```

Responsibilities
Communication entre librarie Graphic et librairie de jeu

Interfaces
SFML
NCURSES

Interfaces
NIBBLER
PACMAN

_interface:IDisplayModule

Content:
```
class IGame
{
  public:
    typedef enum {
        UP,
      DOWN,
        LEFT,
        RIGHT,
        NONE,
      } e_direct;

      virtual ~IGame() {};

      virtual std::vector<std::string> getMap() const = 0;

      virtual void setMap() = 0;
      virtual int isPressed() = 0;
      virtual int outPressed() = 0;
      virtual int forwardPressed() = 0;
      virtual int backwardPressed() = 0;
      virtual int leftPressed() = 0;
      virtual int rightPressed() = 0;

      virtual void findPlayer() = 0;
      virtual void movePlayer() = 0;

      virtual int gameMechs() = 0;
      virtual mapHandle *mapEntry() = 0;
}
```

Responsibilities
Interface d'accès aux librairies de jeux.

_object: mapHandle

Content:
```
public:
    mapHandle(const char *path);
    ~mapHandle();

    void restoreSave() {current = save;};
    std::vector<std::string> current;
private:
    std::vector<std::string> save;
    int index;
```

Responsabilities
Gère le chargement, la modification et la restauration de la map

ARCADE