効率的な自動プレイテストでバグを早期に撃退しよう

具体的な方法

まず、バージョン管理、継続的インテグレーションが必須になります。
バージョン管理には Subversion 等のバージョン管理ツールを使用します。
継続的インテグレーションとは、ビルド作業をワンクリックで行えるようにし、
最低でも毎日自動でビルドを行い、実行可能なバイナリとデータを生成するものです。
これには、Hudson 等のツールが使用できます。

なお、生成したバイナリにはどのリビジョンのデータから生成されたかという情報を埋め込んでおく必要があります。
これは、バグが発生した状況を確実に再現するためです。
もし自動ではなく、特定のプログラマが生成したりすると、ローカルの修正が混じることがあり、
あとから完全に同じ状況を再現することができなくなります。
また、一度停止したバージョンのバイナリは、もう自動プレイテストには使えないので、
修正後にすぐ新しいバイナリをビルドして用意する必要があります。

次に、自動プレイ用のコードをゲームプログラムに組み込みます。
これは、可能な限り早めに作っておくことをおすすめします。
場合によっては設計に若干の修正が入ることもあり、
プロジェクト後半になると組み込む余裕がなくなってしまうことがあるからです。
また、この自動プレイテストはスケジュール上のデバッグ期間に入ってから開始するのではなく、
少しでもゲーム部分が動いた時点で開始することをおすすめします。
デバッグ期間に入る前に、この自動プレイテストで停止バグを退治しておくのが理想です。

自動プレイテスト用コードを作る時のポイントは、可能な限り一度の実行時間を短めに設定することです。
これは、実際に停止バグを発見した場合にプログラマの席でもう一度実行することになるため、
実行時間が長いと確認するまでの時間がかかってしまうからです。
特に区切りがないジャンルの場合、セーブロードをうまく使うか、
デバッグ用に途中から開始させたり、強制的に中断させる仕組みが必要になります。
また、一区切りの自動プレイテストを実行するごとにメモリリークチェックも行うべきです。

スレッドを使用している場合は、強制的に同期させる仕組みを用意する必要があるかもしれません。
同じパラメータで開始した時に、全く同じ動作をすることが重要になります。

一区切りの自動プレイを行う前には、必ずランダムシードを初期化します。
この時、初期化に使うランダムシードは当然ながら毎回変化させる必要があります。
初回はランダムにする必要がありますが、それ以降は 1 つずつ増加させるだけで大丈夫です。

ポイントとして、このランダムシードはどこで停止してもわかるように、画面とコンソールに出力しておきます。
出力するのはランダムシードだけではなく、このバイナリとデータのリビジョンも必要です。
フェードを使っていると、フェードアウト中に確認できないことがあるので注意してください。
必須ではありませんが、このテスト開始からの経過時間や、繰り返し回数なども表示しておくと役に立つことがあります。

実際に停止した後は、停止した時点のランダムシードとリビジョンを元にプログラマの席で同じ状況を再現させます。
ほとんどの場合、デバッガ上で同じように停止するため、容易に原因を特定できます。
停止しない場合、未初期化メモリへのアクセス、停止しないレベルのメモリ破壊、
テスト開始時の初期化抜け等の可能性があります。
これらはコード量が少なければそれだけ発見も容易になるため、早期から自動プレイテストを開始することをおすすめします。
途中から再現しないことが増えた場合、おそらくその間に修正したコードが原因だとわかります。

なお、運用の方法としては一人担当プログラマを決めておいたほうが良いでしょう。
停止した時点では、どこのコードが原因なのかわかりにくいため、まずは担当者が停止箇所を確認、
停止箇所とそのエラー内容を確認してその停止箇所を作ったプログラマに連絡することでスムーズに修正が行えます。

サンプルについて

サンプルコードは
http://www.aqualead.co.jp/CEDEC2010/ にて公開をしています。